НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Односторінковий застосунок для корпоративної переписки»

Виконав:
студент IV курсу, групи ІТ-51 Абрамов Кирило Андрійович _____

Керівник:
Інженер 1-ї категорії, Шинкевич Микола Костянтинович _____

Рецензент:
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2019 рік

Пояснювальна записка
до дипломного проекту
на тему: «Односторінковий застосунок для корпоративної переписки»

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студенту
Абрамову Кирилу Андрійовичу

1.Тема проекту «Односторінковий застосунок для корпоративної переписки», керівник проекту Шинкевич Микола Костянтинович, інженер 1-ї категорії кафедри АУТС,, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом проекту 18.05.2019

3. Вихідні дані до проекту
_____
_____

4. Зміст пояснювальної записки
_____
_____

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
_____
_____-
_____

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка

Студент К.А. Абрамов

Керівник проекту М.К. Шинкевич

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
	А4	КПІ.ІП-5201. ХХХХХХ.02.81	Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud. Технічне завдання		
	А4	КПІ.ІП-5201. ХХХХХХ.02.81	Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud.Пояснювальна записка		
	А4	КПІ.ІП-5201. ХХХХХХ.03.81	Програма та методика тестування.		
	А4	КПІ.ІП-5201. ХХХХХХ.04.51	Керівництво користувача.		
	А4	КПІ.ІП-5201. ХХХХХХ.04.51	Керівництво адміністратора.		

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 1 таблицю, 3 додатків та 16 джерел – загалом 64 сторінки.

Об`єкт дослідження: односторінкові веб системи, що використовуються для корпоративної переписки.

Мета дипломного проекту: створити легко розширюваний односторінковий веб застосунок, здатний до швидкої та зручної комунікації між користувачами.

У першому розділі було проведено аналіз рішень існуючих фреймворків, їх актуальність на сьогоднішній час та вибір фреймворка для застосунка.

У другому розділі було проведено аналіз рішень існуючих веб-застосунків та їх порівняння, вибір метода тестування застосунка.

У третьому розділі було проведено аналіз існуючих архітектур, порівняння їх між та вибір архітектури для нашого застосунку.

У четвертому розділі було проведено введення до користування веб-застосунком, показан інтерфейс та взаємодія користувача з інтерфейсом.

КЛЮЧОВІ СЛОВА: СИСТЕМА, ОДНОСТОРІНКОВИЙ ВЕБ-ЗАСТОСУНОК

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 1 table, 3 applications and 16 sources - a total of 64 pages.

The object of study: single page applications used for corporate correspondence.

The aim of the diploma project: creation of easy-to-expand, sigle page application capable of fast and convenient communication between users.

In the first section, the analysis of the decisions of existing frameworks, their relevance for today and the choice of framework for the application was conducted.

In the second section, an analysis of the decisions of existing web applications and their comparison, and the choice of the method of testing the application was conducted.

In the third section, an analysis of existing architectures, their comparison between architecture choices and our application was conducted.

The fourth section introduced the introduction of the web application, shows the interface and user interaction with the interface.

KEYWORDS: SYSTEM, SINGLE PAGE APPLICATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 14

ВСТУП 15

1АНАЛІЗ РІШЕНЬ ДЛЯ РОЗРОБКИ ОДНОСТОРІНКОВИХ ВЕБ-ЗАСТОСУНКІВ ..... ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.

1.1 JAVASCRIPT ФРЕЙМВОРКИ .....**Ошибка! Закладка не определена.**

ВИСНОВОК ДО РОЗДІЛУ 21

2ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 22

2.1 ОСНОВНІ МЕТОДИ РЕАЛІЗАЦІЇ ЧАТІВ 22

2.2 ОСНОВНІ МЕТОДИ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ 24

2.3 ВИБІР ДОДАТКОВИХ ТЕХНОЛОГІЙ, ЯКІ ПРИСКОРЮЮТЬ РОЗРОБКУ ВЕБ-ЗАСТОСУНКА 28

ВИСНОВОК ДО РОЗДІЛУ 35

ЗРОЗРОБКА АРХІТЕКТУРИ ЗАСТОСУНКА ..... ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.

3.1 ОПИС АРХІТЕКТУРИ REACT + REDUX ЗАСТОСУНКА**Ошибка! Закладка не определена.**

3.2 ІСНУЮЧІ АРХІТЕКТУРИ ОДНОСТОРІНКОВИХ ВЕБ-ЗАСТОСУНКІВ 38

3.3 АРХІТЕКТУРА ВЕБ-ЗАСТОСУНКА 48

3.4 МОДУЛІ ЗБЕРЕЖЕННЯ ТА ОБРОБКИ ДАНИХ 51

3.5 ДІАГРАМА ПРЕЦЕДЕНТІВ 53

ВИСНОВОК ДО РОЗДІЛУ 56

4РЕАЛІЗАЦІЯ ОДНОСТОРІНКОВОГО ВЕБ-ЗАСТОСУНКА 57
4.1 НАЛАШТУВАННЯ НЕОБХІДНИХ МОДУЛІВ ДЛЯ ПОЧАТКУ РОБОТИ З ЗАСТОСУНКОМ 57
4.1.1 Встановлення NodeJS 57
4.2 ОПИС РОБОТИ ЗАСТОСУНКУ 57
ВИСНОВОК ДО РОЗДІЛУ 61
ВИСНОВКИ 62
ПЕРЕЛІК ПОСИЛАНЬ 63
ДОДАТОК А 65
ДОДАТОК Б 66
ДОДАТОК В 67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JavaScript – мова програмування для веб-застосунків[13].

React – бібліотека JavaScript для розробки зручного UI/UX інтерфейсу.

Redux – стейт менеджер React + Redux застосунку[8].

HTML – стандартна мова розмітки документів.

API – набір інтерфейсів з якою взаємодіє клієнт.

Mocha – фреймворк для тестування застосунків[9].

Action Creator – функція за допомогою якої можна обновити Store.

Redux saga – бібліотека для зручної асинхронної взаємодії з API[3].

Redux actions – бібліотека для зручності використання Action Creators[6].

Firebase – відповідає за роль back-end у застосунку, там знаходиться база даних та API[14].

Store – глобальний стейт у React + Redux застосунку[8].

ВСТУП

В даний час за допомогою інтернету доступ до різного роду інформації став простіше і швидше. Інтернет з кожним днем охоплює все більше сфер життєдіяльності людини. Доступ в інтернет для користувачів комп'ютерів і мобільних пристроїв здійснюється при допомогою спеціальної програми, яка отримує дані з серверів, знаходяться в мережі інтернет. У більшості випадків такою програмою є браузер - прикладне програмне забезпечення, призначене для перегляду веб-сторінок, комп'ютерних файлів і каталогів, змісту веб документів, управління веб-додатками, а так само для вирішення інших задач. Функціональні можливості браузерів постійно розширюються і поліпшуються завдяки конкуренції між їхніми розробниками і високим темпом розвитку і впровадження інформаційних технологій. Це робить браузери дуже потужним програмним забезпеченням, здатним на роботу зі багатьма видами медіа-контенту. Саме тому все більша увага приділяється розробці веб-додатків. такі програми можуть працювати на пристроях під управлінням будь-якої операційної системи, в якій встановлений браузер, що робить їх більш зручними для використання. Така кроссплатформенність дозволяє охопити більшу кількість користувачів, так як для роботи не потрібне встановлення додаткового програмного забезпечення.

Практично всі компанії нарівні з електронною поштою і стільниковим зв'язком використовують в альтернативні канали зв'язку для вирішення миттєвих робочих питань. Це можуть бути додатки для відеозв'язку, месенджери, соціальний мережі.

1 АНАЛІЗ РІШЕНЬ ДЛЯ РОЗРОБКИ ОДНОСТОРІНКОВИХ ВЕБ-ЗАСТОСУНКІВ

1.1 Javascript фреймворки

Фронтенд - одне з найбільш динамічно розвиваються напрямків сучасної розробки. Не дивно, що він обріс великою кількістю інструментів, бібліотек і фреймворків, покликаних допомогти в роботі. Допоможе визначитися цей огляд, в якому проаналізовано три аспекти: чому розробники вибирають React, Angular або Vue, наскільки зручно використовувати кожен з них, і аналіз їх затребуваності у роботодавців.

Для початку варто порівняти кількість опублікованих вакансій на різних ресурсах, рисунок 1.1.1.

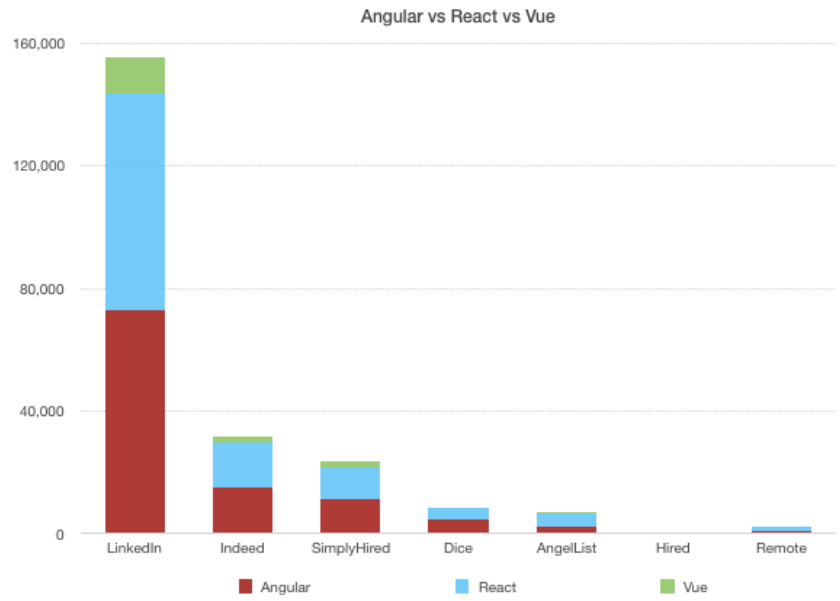

Рисунок 1.1.1 - Кількість опублікованих вакансій

Вибір ресурсів заснований на:

- Linkedin. Зібрані вакансії розробників з усього світу.
- Indeed, SimplyHired і Dice. 3 найбільших сайту для пошуку роботи в США.

- AngelList. Пошук роботи в стартапи (дозволяє зрозуміти, які технології затребувані).
- Hired. На цьому майданчику великі IT-компанії шукають талановитих розробників.
- Віддалена робота. Оголошення про пошук роботи з сайтів indeed.com і remote.co додані для повноти списку.

За наведеними даними видно, що більшість розробників готові працювати з React і Angular. І якщо це не дивно для React, частота використання якого тільки зростала останні кілька років, то з Angular історія інша. The State of JavaScript хоч і з застереженням, але вважають, що популярність Angular падає.

Однак виходячи тільки з розміщених вакансій, слід вивчати і React, і Angular, щоб розширити рамки пошуку роботи.

Наступний пункт аналізу стосується використання розробниками React, Angular і Vue. На рисунку 1.1.2 статистика їх завантажень за 2 роки згідно з даними з NPMtrends:

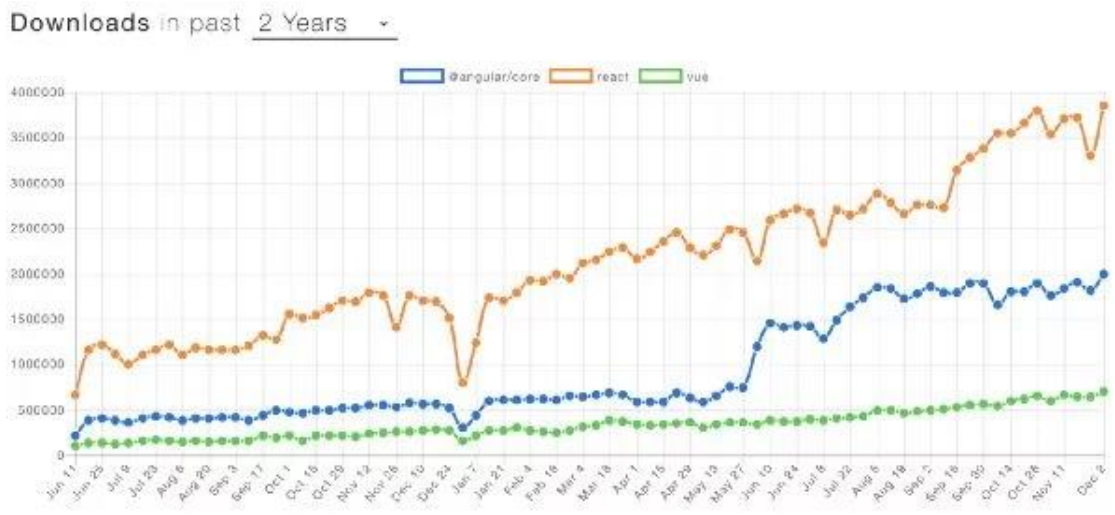

Рисунок 1.1.2 - Статистика завантажень фреймворків

Виходячи з даних, кількість завантажень росте. Однак статистика використання Vue йде врозріз з кількістю поставлених йому зірок.

Незважаючи на те, що помітна тенденція зниження використання React і Angular, вони все ще займають лідируючі місця. Vue використовують рідше, але уповільнення темпів зростання у використанні React і Angular може вказувати на перехід користувачів до цього фреймворку. Що стосується документації, то корисної інформації по React більше, ніж за Angular або Vue.

Нижче на рисунку 1.1.3 наведені дані ресурсу StackOverflow про фреймворк, бібліотеках і інструментах з якими розробники хотіли б попрацювати:

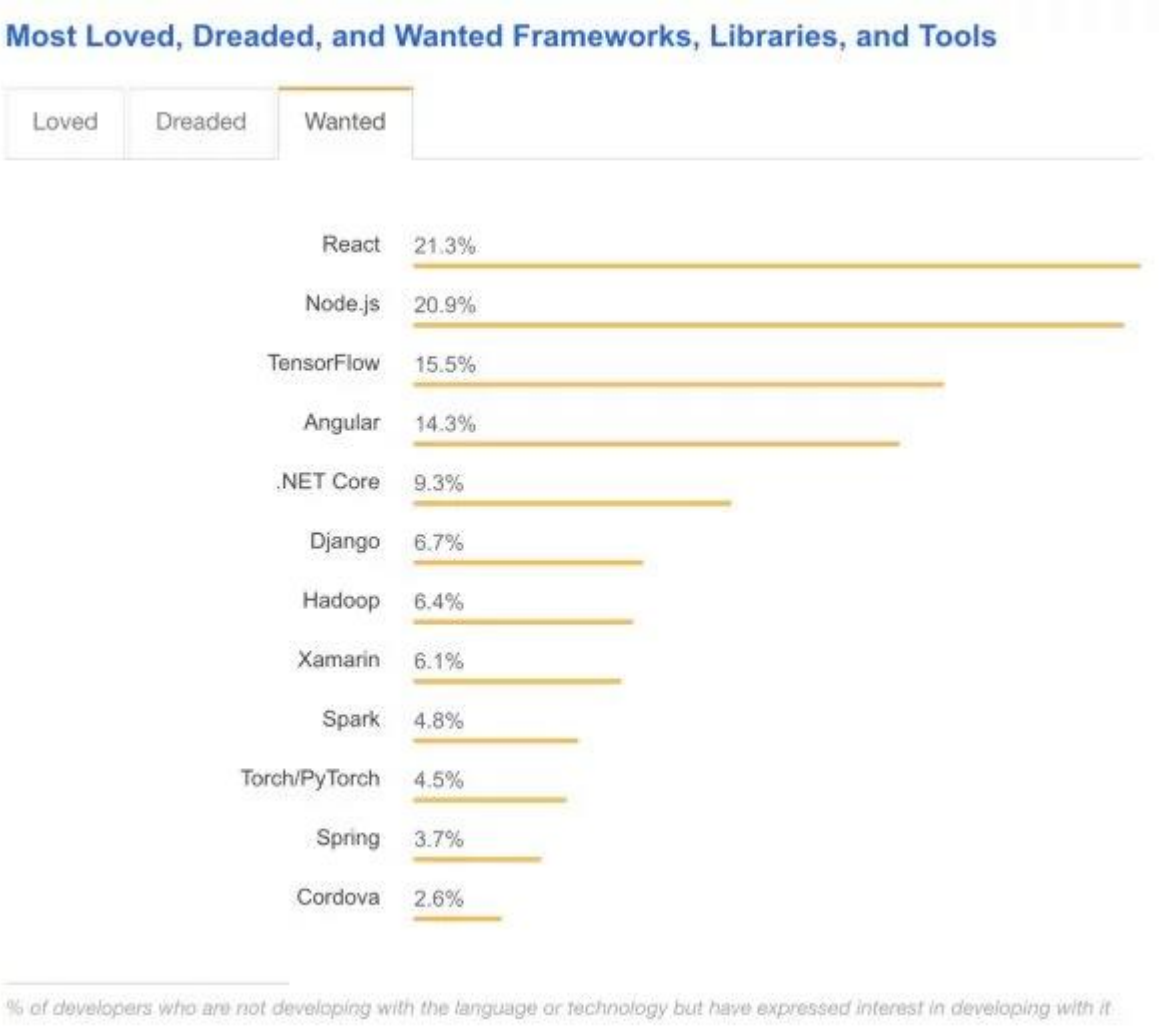

Рисунок 1.3 - Інструменти з якими розробники хотіли б попрацювати

Беручи до уваги отримані дані, з’ясуємо основні переваги і недоліки кожного фронт-енд фреймворка і, таким чином, виберемо найбільш підходящий з них для нашого застосунку.

Плюси і мінуси Angular 5

Angular - це фреймворк MVVM JavaScript, заснований в 2009 році, який ідеально підходить для створення інтерактивних веб-додатків.

Переваги Angular 5:

- Нові функції, такі як поліпшений RXJS, прискорена компіляція (менше 3 секунд) і новий лаунчер HttpClient;
- Детальна документація, яка дозволяє кожному розробнику отримати всю необхідну інформацію без звернення за допомогою до колег. Проте, навчання вимагає чималого часу;
- Двостороння прив'язка даних, яка забезпечує чудову поведінку додатка, що мінімізує ризики можливих помилок;
- MVVM (Model-View-ViewModel), яка дозволяє розробникам окремо працювати в одному розділі програми з використанням одного і того ж набору даних;
- Впровадження залежностей функцій пов'язаних з компонентами з модулями і модульності в цілому.

Недоліки Angular 5:

- Складний синтаксис, який виходить від першої версії Angular. Проте, Angular 5 використовує TypeScript 2.4, який вивчити не так вже й складно;
- Проблеми з міграцією, які можуть виникнути при переході від старої версії до нової;

Компанії, які використовують Angular 5: Upwork, Freelancer, Udemy, YouTube, Paypal, Nike, Google, Telegram, Weather, iStockphoto, AWS, Crunchbase.

Плюси і мінуси ReactJS

ReactJS - це бібліотека JavaScript, вихідний код якої був відкритий Facebook в 2013 році. Цей фреймворк відмінно підходить для створення величезних веб-додатків, де дані можуть змінюватися на регулярній основі. Також разом з нею використовуються різні бібліотеки стану.

Переваги ReactJS:

- Легкий у вивченні. React набагато легше вчиться зважаючи на простоту його синтаксису. Інженери просто повинні згадати свої навички написання HTML і все на цьому. Немає потреби в глибокому вивченні TypeScript, як у випадку з Angular;
- Високий рівень гнучкості і максимальна чуйність;
- Віртуальний DOM (document object model), яка дозволяє впорядковувати документи форматів HTML, XHTML або XML в дерево, яке найкраще підходить веб-браузерів для аналізу різних елементів веб-додатки;
- У поєднанні з ES6 / 7 ReactJS може легко працювати при високих навантаженнях;
- Зв'язування даних від великих до менших. Це означає такий потік даних, при якому дочірні елементи не можуть впливати на батьківські дані;
- 100% -а JavaScript-бібліотека з відкритим вихідним кодом, яка отримує безліч щоденних оновлень і поліпшень відповідно до відгуками розробників по всьому світу;
- Неймовірно легка вага, так як дані, які виконуються на стороні користувача, можуть легко бути представлені на стороні сервера в той же самий час;
- Міграція між версіями, як правило, дуже проста. Також Facebook надає «codemods» для автоматизації більшої частини цього процесу.

Недоліки ReactJS:

- Недостача офіційної документації. Надзвичайно швидка розробка ReactJS не залишає місця для правильної документації, яка зараз трохи хаотична, так як багато розробників вносять в неї індивідуальні зміни без будь-якого систематичного підходу;
- React не має чіткої мети. Це означає, що розробники, іноді, мають занадто великий вибір;
- Довгий час для освоєння. React JS вимагає глибокого розуміння того, як інтегрувати користувальницький інтерфейс в структуру MVC.

Компанії, які використовують ReactJS: Facebook, Instagram, Netflix, New York Times, Yahoo, Khan Academy, Whatsapp, Codecademy, Dropbox, Airbnb, Asana, Atlassian, Intercom, Microsoft.

Плюси і мінуси Vue.js

Vue.js - це фреймворк JavaScript, запущений в 2013 році, який ідеально підходить для створення адаптуються користувальницьких інтерфейсів і складних односторінкових додатків.

Переваги Vue.js:

- Посилений HTML. Це означає, що Vue.js має багато схожих з Angular характеристик. Це може допомогти оптимізувати обробку HTML-блоків з використанням різних компонентів;
- Детальна документація. Vue.js має дуже хорошу документацію, яка може збільшити швидкість навчання розробників і заощадити багато часу на розробку програми з використанням базових знань HTML і JavaScript;
- Адаптивність. Vue.js забезпечує швидкий період переходу від інших фреймворків до Vue.js на увазі його схожості з Angular і React з точки зору дизайну і архітектури;
- Чудова інтеграція. Vue.js можна використовувати як для створення односторінкових додатків, так і для більш складних веб-інтерфейсів додатків. Найважливіше, що невеликі інтерактивні частини можна легко інтегрувати в існуючу інфраструктуру, не надаючи при цьому негативного впливу на всю систему;
- Велике масштабування. Vue.js допомагає розробляти досить великі шаблони для багаторазового використання, які можна розробити без витрачання величезної кількості часу на увазі простий структури;
- Крихітний розмір. Vue.js може важити близько 20 КБ і при цьому зберігати свою свою швидкість і гнучкість, що дозволяє досягти набагато більш високої продуктивності, в порівнянні з іншими фреймворками.

Недоліки Vue.js:

- Недостача ресурсів. Vue.js як і раніше має досить невелику частку ринку в порівнянні з React або Angular. Це означає, що обмін знаннями в рамках фреймворка все ще формується;
- Ризик надмірної гнучкості. Іноді у Vue.js можуть виникати проблеми при інтеграції в величезні проекти, а досвіду про можливі рішення до сих пір немає. Але вони обов'язково з'являться найближчим часом;
- Відсутність повної англомовної документації. Це призводить до деяких складнощів на різних етапах розробки. Проте, все більше і більше матеріалів перекладаються англійською мовою.

Vue все ще дуже новий і швидко розвивається. Вона ще не має широкої підтримки своїх колег, оскільки вона не настільки популярна, як React або Angular. Він був створений китайсько-американським, і більшість користувачів не є англомовними. Більша частина кодування написана китайською мовою, що ускладнює речі для англомовних розробників. На жаль, оскільки вона швидко розвивається, багато навчальних посібників, які ви знайдете в Інтернеті, можуть бути застарілими. Це означає, що якщо ви застрягли, вам може знадобитися більше часу, щоб знайти рішення. Відповідним питанням може бути те, що ще немає багато бібліотек / розширень.

Компанії, які використовують Vue.js: Xiaomi, Alibaba, WizzAir, EuroNews, Grammarly, Gitlab and Laracasts, Adobe, Behance, Codeship, Reuters.

Висновок до розділу

У цьому розділі було описано та проаналізовано фреймворки, які можуть використовуватися у розробці веб-застосунка. Було виділено подібні фреймворки у області односторінкових веб-застосунків та порівняно між собою. Для розробки був обраний найкращій фреймворк по статистиці розробників, з метою створити веб-застосунок, який буде швидко масштабуватися та підтримуватися різними командами людей.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

2.1 Основні методи реалізації чатів

Було проведено пошук і аналіз існуючих чат-сервісів для корпоративних користувачів. На ринку існує безліч програмних продуктів для комунікацій усередині робочої команди. Нижче розглянуті деякі з них:

− Staply. Мессенджер з проектним функціоналом. Дозволяє скласти ієрархію груп; створювати завдання, задавати терміни, важливість і відповідальність в робочих групах; здійснювати контроль над групами;

− Slack. Корпоративний месенджер з підтримкою інтеграції з десятками сторонніх сервісів. Об'єднує в одному вікні обговорення в загальних темах (каналах), приватних групах і особистих повідомленнях. Дозволяє здійснювати відеодзвінки; зберігати повідомлення без доступу до мережі;

− HipChat. Являє собою груповий чат, відеочат для команд і компаній з можливостями обміну файлами і демонстрацією екрану;

− Bitrix24. Система управління внутрішнім інформаційним ресурсом компанії для колективної роботи над завданнями, проектами і документами, для ефективних внутрішніх комунікацій. Дозволяє контролювати, делегувати, оцінювати завдання; створювати резервні копії в декількох місцях; працювати в декількох кімнатах одночасно.

На основі функцій аналогів і поставлених завдань можна виділити наступні критерії для порівняння:

- інтеграції зі сторонніми сервісами;
- створення ієрархії груп;
- розподіл керуючим учасників за ролями;
- можливість контролю керівником над групами;
- резервне копіювання в декількох місцях
- робота в декількох кімнатах одночасно;

- настроюється оформлення;
- інструменти
- оффлайн-повідомлення

Таблиця 2.1 - Порівняльна характеристика розглянутих аналогів

Критерій	Аналоги				Примітка
	Staply	Slack	HipChat	Bitrix24	
Інтеграція зі стороннім и сервісами	-	Dropbox, Google Drive, Github, Google Docs, Google Hangouts, Twitter та інші.	Github, MailChimp, Heroku, та інші.	MS SharePoint, MS Exchange Server, MS Outlook, Продукти Apple и Google.	Дозволяє користувачам відслідковувати прогрес у різних проектах за допомогою однієї платформи
Створення ієрархії груп	+	-	-	-	Дозволяє рівномірно розподілити завдання
Розподіл керуючим учасників за ролями	-	-	-	-	Дозволяє ефективно розподілити завдання
Можливість контролю керуючим над проектом	+	+	+	+	Забезпечує контролювання виконання завдань
Резервне копіювання	+/-	-	-	+	Забезпечує збереження даних

Продовження до таблиці 2.1

Робота в кількох кімнатах одночасно	+	-	-	+	Дозволяє вирішувати кілька завдань одночасно
Настроюється оформлення	-	-	-	-	Дозволяє індивідуалізувати інтерфейс
Інструменти	Блокнот, Задачі	-	-	Щоденник, планувальник подій, календар, графік відсутностей, зборів та планерки	Дозволяють оперативно виконувати завдання
Оффлайн-повідомлення	+	+	+	-	Дозволяє працювати з інформацією без мережі

2.2 Основні методи тестування веб-застосунків

Mocha - багатофункціональний тестовий JavaScript фреймворк, який працює на Node.js і в браузері, з підтримкою зручного асинхронного тестування. Тести Мокка забезпечують точну і гнучку звітність і виконуються послідовно, обробляючи не перехоплені виключення. Цей фреймворк є потужним і одним з кращих інструментів для тестування. Розглянемо його основні особливості:

- підтримка тестування в браузері;
- рядкова diff (утиліта, що показує різницю між двома файлами);
- підтримка JavaScript API для запуску тестів;

- власний код на виході для підтримки безперервної інтеграції;
- відображення перехоплених винятків для правильних результатів тесту;
- асинхронна підтримка тайм-ауту тесту;
- тест конкретних тайм-аутів;
- підтримка повідомлень;
- маркування повільних тестів;
- авто-вихід (для запобігання "зависання" активного циклу роботи програми);
- проста мета-генерація тестових наборів і випадків;
- інтерактивні заголовки наборів для фільтрації виконання тесту;
- підтримка декількох викликів функції done ();
- довільна підтримка спрощеного мови (CoffeeScript і тд.).

Mocha, завдяки своїй гнучкості, дозволяє використовувати різні бібліотеки тестування, які забезпечують різний підхід до реалізації тестування. Перерахуємо кілька таких бібліотек:

- should.js (behaviour-driven development стиль тестування);
- expect.js (стиль тестування - очікування / твердження);
- chai.js (стиль тестування - очікування / твердження);
- better-assert

Можливості Mocha більш докладно.

Асинхронне тестування - так як обраний фреймворк побудований на базі мови JavaScript, то він дозволяє проводити асинхронне тестування. Асинхронні тести на Mocha полягають в тому, що в функцію зворотного виклику передається додатковий параметр "done", який є функцією і викликається в випадку успішного проходження тесту. Реалізацію функції describe(), яка описує як буде поводитися новий екземпляр класу User, можна побачити на рисунку 2.2.1.

Всі функції хуки (before(), after(), beforeEach(), afterEach()), забезпечують різні умови тестування (виконання частини коду, до або після всіх тестів в блоці, перед або після кожного тесту в блоці), також підтримують асинхронну модель

поведінки. завдяки додатковому параметру done забезпечується проста підтримка асинхронного тестування.

```
describe('User', function() {
  describe('#save()', function() {
    it('should save without error', function(done) {
      var user = new User('Luna');
      user.save(done);
    });
  });
});
```

Рисунок 2.2.1 - Приклад функції describe () з додатковим параметром done

2) Синхронне тестування - воно здійснюється за допомогою запуску функції зворотного виклику, яка забезпечує перехід до наступного тесту, в разі успішного завершення попереднього тесту. синхронне тестування виглядає також як і асинхронне. Різниця в тому, що в синхронному тестуванні відсутня оператор done() через його непотрібність.

3) Логування - Mocha підтримує стандартну JavaScript-функцію - console.log(). Ця функція забезпечує висновок додаткової інформації в консоль під час роботи скриптів тестування, дозволяючи тестувальника спостерігати за ходом виконання окремих частин функцій тестування.

4) Mocha дозволяє запускати тестування в різних режимах. за замовчуванням Мокка запускає виконання тестів один раз, завершує свою роботу і знову чекає команди запуску. Для того щоб відстежувати зміни в коді і запускати тестування автоматично, необхідно використовувати наступну команду - "$ mocha -watch". Завдяки автоматичному режиму, розробник позбавляється від запуску тестування після кожного внесеного зміни в коді.

5) Тестування окремих частин даних - функції тестування дозволяють здійснювати тестування визначених наборів даних, використовуючи функцію

only(). На рисунку 2.2.2 наведено приклад коду, який дозволяє вибрати певні типи даних серед інших значень і протестувати їх окремо. Такий гнучкий підхід дозволяє виробляти тестування окремих частин додатків, які не зачіпаючи інші дані.

```
describe('Array', function() {
  describe.only('#indexOf()', function() {
    // ...
  });
});
```

Рисунок 2.2.2 - Приклад використання функції only()

6) Пропуск тестування окремих частин даних - функція skip() дуже схожа за способом роботи на функцію only(). Функція skip() дозволяє пропускати окремі частини даних, без їх проведення над ними тестів. Така поведінка іноді необхідно. На рисунку 2.2.3 наведено код, який дозволяє пропустити тестування певних частин даних. Така можливість іноді необхідна, у випадках, якщо тестувальник не знає як повинні змінюватися деякі частини тестованих даних.

```
describe('Array', function() {
  describe.skip('#indexOf()', function() {
    // ...
  });
});
```

Рисунок 2.2.3 - Використання функції skip() для пропуску тестування даних

7) Мета-генерація тестів - так як Mocha використовує виклики функціональних виразів для визначення специфікацій тестів, то це дозволяє створювати тести досить просто. При створенні тестів досить використовувати простий синтаксис мови JavaScript для досягнення аналогічної функціональності параметеризованих тестів в різних частинах коду, призначеного для проведення тестування.

8) TDD і BDD в Mocha - Mocha дозволяє розробнику вибирати стиль мови програмування, спеціалізованого для конкретної області застосування (DSL - Domain Specific Language). Завдяки DSL в Mocha можна використовувати техніки BDD (Behavior-driven development) і TDD (test-driven development), що робить Мокка універсальним інструментом для управління розробкою тестів. TDD - це техніка розробки програмного забезпечення, яка ґрунтується на повторенні дуже коротких циклів розробки: спочатку пишеться тест, покриває бажану зміну, потім пишеться код, який дозволить пройти тест, і під кінець проводиться рефакторинг (процес зміни внутрішньої структури програми для полегшення читаності програмного коду) нового коду до відповідних стандартів. Техніка BDD тестування заснована на техніці TDD тестування з деякими змінами, які дозволяють більш ефективно працювати тестувальників.

2.3 Вибір додаткових технологій, які прискорюють розробку веб-застосунку

ReactJS + Redux - це інструмент управління як станом даних, так і станом інтерфейсу в JavaScript-додатках. Він підходить для односторінкових додатків, в яких управління станом може з часом стає складним. Redux не пов'язаний з якимось певним фреймворком, і хоча розроблявся для React, може використовуватися з Angular або jQuery.

Елементи - це JavaScript об'єкти, які представляють собою HTML-елементи. Їх не існує в браузері, вони описують DOM-елементи такі як div, h1 або button.

Компоненти - це елементи React, написані розробником. зазвичай це частини призначеного для користувача інтерфейсу, які містять свою структуру і функціональність. Наприклад, такі як NavBar, LikeButton, або ImageUploader.

Властивості - опції компонента. Вони надаються в якості аргументів компонента і виглядають так само, як атрибути HTML.

Стан - це спеціальний об'єкт всередині компонента. він зберігає дані, які можуть змінятися з часом.

Композиція - комбінування менших компонентів при формуванні більшого.

Особливості:

1) Одностороння передача даних - властивості передаються в рендерер компонента, як властивості html тега. Компонент не може безпосередньо змінювати властивості. Однак компонент може мати внутрішній стан. Це об'єкт this.state, доступний всередині самого компонента;

2) Віртуальний DOM - React підтримує віртуальний DOM (Document Object Model). це дозволяє бібліотеці визначити, які частини DOM змінилися в порівнянні із збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера;

3) JSX - компоненти React зазвичай написані на JSX - розширення синтаксису JavaScript, що нагадує XML, яке дозволяє використовувати синтаксис HTML тегів для рендеринга компонентів Код написаний на JSX компілюється в виклики методів бібліотеки React.

Redux - це JavaScript бібліотека з відкритим вихідним кодом, призначена для управління станом додатки. Він в основному використовується спільно з React для створення користувацьких інтерфейсів. Добре підходить для односторінкових додатків, в яких управління станом може з часом стає складним.

Redux пропонує зберігати все стан додатки в одному місці, котре називається «store» («сховище»). Компоненти «відправляють» зміна стану в сховище, а не безпосередньо до інших компонентів.

Компоненти, які повинні бути в курсі цих змін, «підписуються» на сховище. Сховище може розглядатися як «посередник» у всіх зміни стану в додатку. З Redux компоненти не зв'язуються один з одним безпосередньо, всі зміни повинні пройти через єдиний джерело істини, через сховище. С Redux всі компоненти отримують своє стан зі сховища. Також ясно, куди компонент повинен відправити інформацію про зміну стану - знову ж в сховище. компонент тільки ініціює зміну і не піклується про інших компонентах, які повинні отримати цю зміну. Таким чином, Redux робить потік даних більш зрозумілим.

Три основних принципи Redux:

1) Redux використовує тільки одне сховище для всього стану додатки. Оскільки стан знаходиться в одному місці, його називає єдиним джерелом істини. Структура даних сховища повністю залежить від розробника, але для реального застосування це, як правило, об'єкт з декількома рівнями укладення.

2) Згідно з документацією Redux, «Єдиний спосіб змінити стан - передати екшен - об'єкт, що описує, що сталося ». це значить, що програма не може безпосередньо змінити стан. Замість цього, необхідно передати «action», щоб висловити намір змінити стан в сховище.

3) Redux не дозволяє змінювати стан безпосередньо. Замість цього екшен описує, які зміни необхідно зробити. Редьюсери (reducers) - це функції, які обробляють екшени і можуть вносити зміни в стан.

Редьюсери повинні бути реалізовані як "чисті" функції (pure functions), термін, що описує функції, що задовольняють наступним умовами:

- Вони не повинні робити зовнішніх викликів по мережі або базі даних;
- Вони повертають значення, залежне тільки від переданих параметрів;
- Їхні аргументи є незмінними, тобто функції не повинні їх змінювати;
- Виклик чистої функції з тими ж аргументами завжди повертає однаковий результат.

Ці функції називають "чистими", тому що вони не роблять нічого, тільки повертають значення, залежне від параметрів. Вони не залежать від будь-якої з частин системи.

Серед найкращих і найбільш популярних BaaSs можна виділити Firebase від компанії Google. По суті, Firebase є безумовно приголомшливим у виконанні, реалізації та експлуатації.

У зв'язку з несподіваним рішенням Facebook закрити Parse, багато розробники задалися питанням, що використовувати замість нього. Сьогодні практично неможливо уявити повністю автономне додаток, яке було б корисне всім. У зв'язку з цим, iOS розробники в своїй роботі користуються інструментами і ресурсами, надані Apple для доступу до даних. Backend-as-a-service, або скорочено BaaS є приголомшливим інструментом для розробників.

Firebase служить базою даних, яка змінюється в реальному часі і зберігає дані в JSON. Будь-які зміни в базі даних відразу синхронізуються між усіма клієнтами, або девайсами, які використовують одну і ту ж базу даних. Іншими словами, оновлення в Firebase відбуваються миттєво.

Разом зі сховищем, Firebase також надає призначену для користувача аутентифікацію, і тому всі дані передаються через захищене з'єднання SSL. Ми можемо вибрати будь-яку комбінацію email і пароля для аутентифікації, будь то Facebook, Twitter, GitHub, Google, або щось інше.

В добавку до iOS SDK, у Firebase є SDK для Android і JavaScript. Всі платформи можуть використовувати одну базу даних.

Складно уявити що Firebase з усіма цими функціями бюджетне рішення.

Додаткові бібліотеки для прискорення розробки застосунка:

− Semantic UI

Сучасний front-end фреймворк, який працює на основі LESS і jQuery. Він має гладкий, тонкий і плоский дизайн, який забезпечує легкий користувальницький досвід.

Згідно з веб-сайтом Semantic UI, метою є рамки розширення можливостей дизайнерів і розробників "шляхом створення мови для спільного використання інтерфейсу користувача". Вони роблять це шляхом використання семантичної, описової мови для своїх класів і умов імен. Замість використання абревіатур, як це роблять інші рамки, він використовує реальні слова, наближені до простого англійського.

Майже кожен компонент має типи, стани та варіації. Наприклад, деякі типи компонентів кнопок включають: стандартну кнопку, кнопку з піктограмою, анімовані кнопки, а кнопка може бути в активному, вимкненому або завантаженому стані. Нарешті, кнопка може відрізнятися за розміром і кольором, і може бути відформатована як основна, соціальна, перемикальна і багато іншого. Ця схема надає велику гнучкість у зовнішньому вигляді компонента.

Semantic UI не тільки змістовний і добре структурований з точки зору іменування його класів, але і в іменуванні, визначенні та описі його компонентів. Ця структура набагато більш семантична в порівнянні з тією, що знайдена у Bootstrap або Foundation.

Друга унікальна річ у Semantic UI полягає в тому, що вона надає деякі ексклюзивні функції та компоненти, не присутні в інших рамках. Наприклад, Feed та Comment в компонентах UI Views або Sidebar і Shape з модулів UI. Крім того, при взаємодії з компонентами Semantic UI ви отримуєте вивід налагодження в реальному часі. Просто відкрийте веб-консоль, і ваші компоненти спілкуватимуться саме з тим, що вони роблять.

Ще одна сила Semantic UI полягає в тому, що він використовує мінімальний і нейтральний стиль, залишаючи налаштування відкритим для вас. Вона включає в себе важливі та корисні речі, залишаючи при цьому додаткові функції, які, напевно, ніколи не використовуються. Крім того, компоненти фреймворку є портативними та автономними, тому ви можете захопити і використовувати тільки ті, які вам потрібні.

Каркас використовує em і rem одиниць для своїх елементів, що робить його повністю чуйним і адаптивним до будь-якого розміру. Вам потрібно лише змінити базовий шрифт, і всі інші елементи будуть відповідно налаштовані.

Нарешті, Semantic UI дуже добре задокументований, і веб-сайт містить багато прикладів для різних компонентів. Крім того, він має стиль керівництво з методиками і вказівки про те, як написати свій код. Все це робить навчання рамками безболісним досвідом.

− Redux Saga

Бібліотека, яка покликана спростити і покращити виконання сайд-ефектів (тобто таких дій, як асинхронні операції, типу завантаження даних і "брудних" дій, типу доступу до браузерних кешу) в React / Redux додатках.

Можна уявити це так, що сага - це як окремий потік в вашому додатку, який відповідає за сайд-ефекти. redux-saga - це redux middleware, що означає, що цей потік може запускатися, зупинятися і скасовуватися з основного додатка за допомогою звичайних redux екшенів, воно має доступ до повного станом redux додатки і також може діспатчіть redux екшени.

Бібліотека використовує концепцію ES6, під назвою генератори, для того, щоб зробити ці асинхронні потоки легкими для читання, написання та тестування. (Якщо ви не знайомі з цим, тут є деякі посилання для ознайомлення) Тим самим, ці асинхронні потоки виглядають, як ваш стандартний синхронний JavaScript код. (На зразок async / await, але генератори мають кілька відмінних можливостей, необхідних нам)

Можливо, ви вже використовували redux-thunk, перед тим як обробляти ваші вибірки даних. На відміну від redux thunk, ви не опиняється в callback пеклі, ви можете легко тестувати ваші асинхронні потоки і ваші екшени залишаються чистими.

У нас є інтерфейс для вилучення деяких призначених для користувача даних з віддаленого сервера при натисканні кнопки, рисунок 2.3.1.

```
class UserComponent extends React.Component {
  ...
  onSomeButtonClicked() {
    const { userId, dispatch } = this.props
    dispatch({type: 'USER_FETCH_REQUESTED', payload: {userId}})
  }
  ...
}
```

Рисунок 2.3.1 - Приклад використання функції з екшеном

Компонент діспатчіт action у вигляді простого об'єкта в Store. Ми створимо сагу, яка слухає все USER_FETCH_REQUESTED екшени і тригере виклики API для отримання призначених для користувача даних.

Redux-saga надає деякі допоміжні ефекти, які переносять внутрішні функції для створення завдань, коли деякі конкретні дії надсилаються до магазину.

Допоміжні функції побудовані поверх API нижчого рівня. У розширеному розділі ми побачимо, як ці функції можуть бути реалізовані.

Перша функція takeEvery є найбільш знайомою і забезпечує поведінку, подібну до redux-thunk.

На відміну від takeEvery, takeLatest дозволяє виконувати лише одну задачу fetchData у будь-який момент. І це буде останнє почате завдання. Якщо попереднє завдання все ще працює, коли запускається інша задача fetchData, попереднє завдання буде автоматично скасовано.

Якщо у вас є кілька саг, які спостерігають за різними діями, ви можете створити декількох спостерігачів з тими вбудованими помічниками, які будуть вести себе так, як коли б їхні fork використовувалися.

У redux-saga Саги реалізовані з використанням функцій генератора. Щоб висловити логіку Saga, ми виводимо звичайні об'єкти JavaScript з генератора. Ми називаємо ці Ефекти Об'єктів. Ефект - це об'єкт, який містить певну інформацію, яка буде інтерпретована проміжним програмним забезпеченням.

Висновок до розділу

У даному розділі були обрані допоміжні бібліотеки, які в подальшому допоможуть нашому затосунку працювати з великим потоком даних, які відповідають за стилістику нашого застосування, роботу з асинхронними запитами і зі стейт менеджером.

Обрана мова, яка найкраще підходить для реалізації поставленної задачі – JavaScript, та бібліотеки React + Redux, в поєднанні з JSX, замість стандартного HTML5, та Semantic UI + CSS Modules, замість стандартної мови CSS.

Обрана мова для тестування, та опис існуючих реалізацій тестування.

Були описані конкурентні мессенджери та їх переваги і недоліки.

3 РОЗРОБКА АРХІТЕКТУРИ ЗАСТОСУНКА

3.1 Опис архітектури React + Redux застосунка

React - це бібліотека view. Вона повинна давати відображення данних, які їй надаються, і не більше того.

React не повинен запитувати дані, якщо їх немає.

Це не є частиною відображення. Компонент повинен просто відображати інтерфейс користувача для доступних даних. Якщо дані відсутні, покажіть стан за замовчуванням або стан помилки залежно від сценарію.

У нас є компонент React, який повинен показувати деталі списку з "id" 6. Компонент не повинен перевіряти, чи не доступні дані. Він повинен відправити action для отримання даних.

Найкращим рішенням буде отримання даних з actions. Компонент піклується про перегляд вмісту HTML на основі наданих даних.

Точка входу до вашої програми - це маршрути. Коли маршрут узгоджується, завантажується компонент, відповідний цьому маршруту. Викликається обробник маршруту для цього маршруту. Отже, обробник маршрутів піклується про відправлення цієї дії, рисунок 3.1.1.

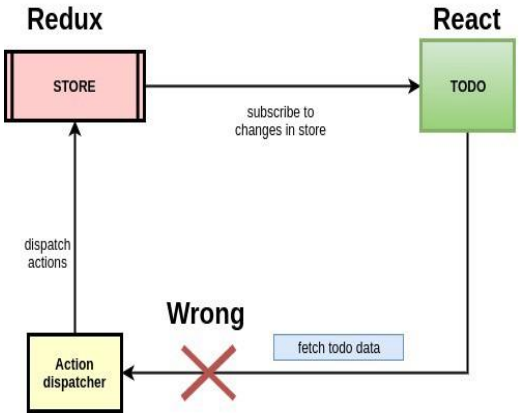

Рисунок 3.1.1 - Флоу React + Redux застосунка

Події з життєвого циклу, якщо вони не використовуються належним чином, можуть призвести до багатьох питань, які можуть бути нелегкими для налагодження. Навіть якщо ви зможете знайти проблему, вам доведеться вносити багато змін. Це може зробити ваш компонент ще більш складним. Ви можете повністю змінити структуру.

На рисунку 3.1.2 зображено правильної та неправильної реалізацію архітектури.

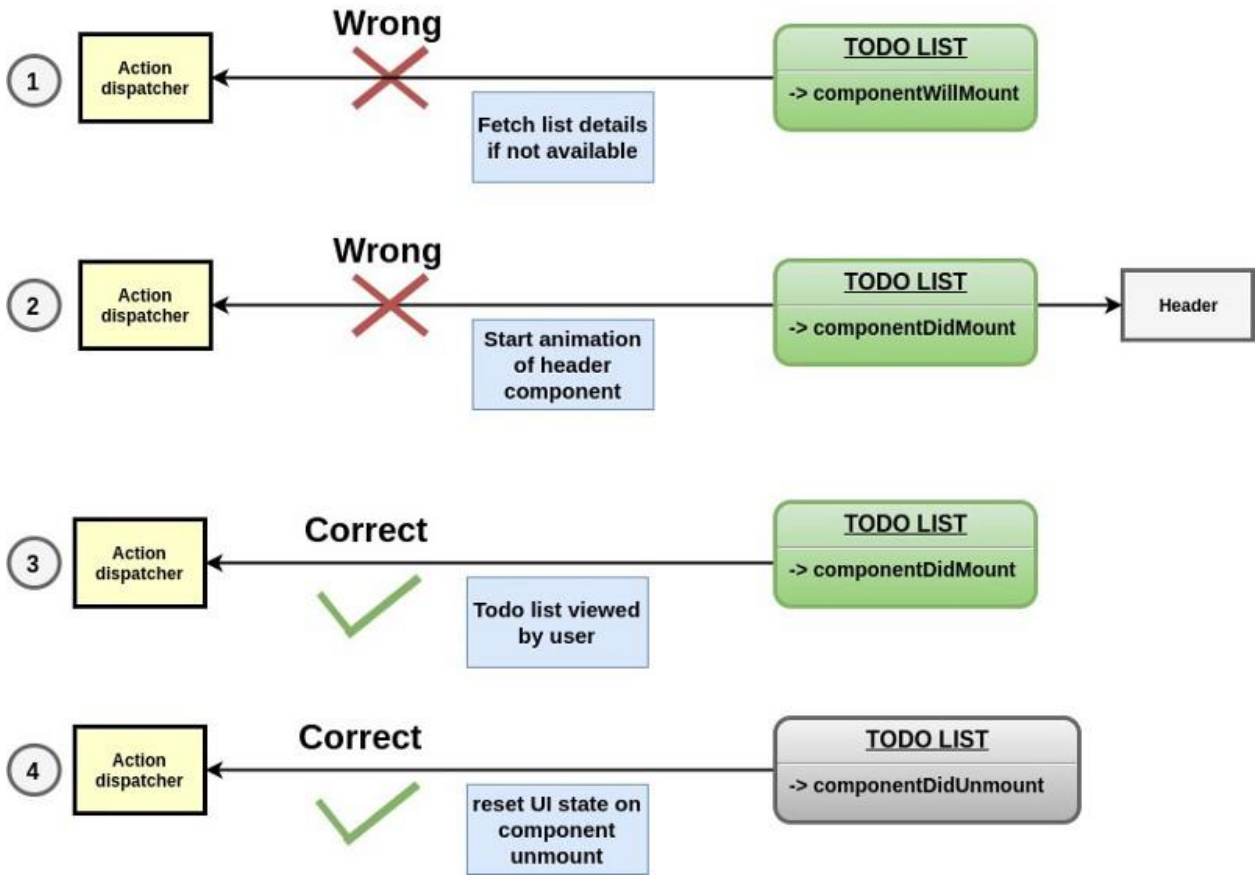

Рисунок 3.1.2 - Порівняння реалізації архітектур застосунків

За бізнес логіку застосунку буде відповідати Redux. Він складається з трьох частин: Actions, Reducers і Store. За мозок застосунка відповідає Actions і Reducers. За сховище даних - Store.

Таким чином, дані застосунка зберігаються в Store redux і представлення в React. Все інше, що буде частиною застосунка, прийде в «мозок». Він вирішить, як функціонуватимуть бізнес логіка та представлення.

Проблеми, які буде вирішувати Redux:

− Бізнес-логіка - включає в себе модель, яка має логіку / алгоритм для визначення стану нашої програми. Вона може включати в себе все, від простих алгебраїчних операцій до набору кроків для виконання певного завдання;

− Обробка маршрутів - буде містити всі речі, які повинні відбутися після того, як ви потрапили на будь-який маршрут. Буде перевіряти правильність маршруту, чи користувач має дозвіл потрапити на цей маршрут, та якщо потрібно буде редіректити його на іншу сторінку, чи доступно достатньо даних для цього маршруту, якщо ні, то буде отримувати дані;

– Виклики API - всі запити AJAX прийдуть сюди. Вони будуть включати в себе алгоритм, за яким повинні відповідати всі AJAX-запити, проблеми керування помилками або підключення до Інтернету, синхронізацію або передачу паралельних численних викликів API, групування декількох API, щоб його можна було повторно використовувати;

– Авторизація - включає всі типи аутентифікації, які будуть частиною застосунка. Наприклад авторизації користувача, перегляд певного інтерфейсу користувача та здійснення певної дії.

3.2 Існуючі архітектури односторінкових веб-застосунків

Atomic design architecture - це методологія, що складається з п'яти різних етапів, що працюють разом для створення систем дизайну інтерфейсів більш продуманим і ієрархічним способом.

П'ять етапів атомного проектування, які можна побачити на рисунку 3.2.1:

– Атоми

– Молекули

– Організми

– Шаблони

– Сторінки

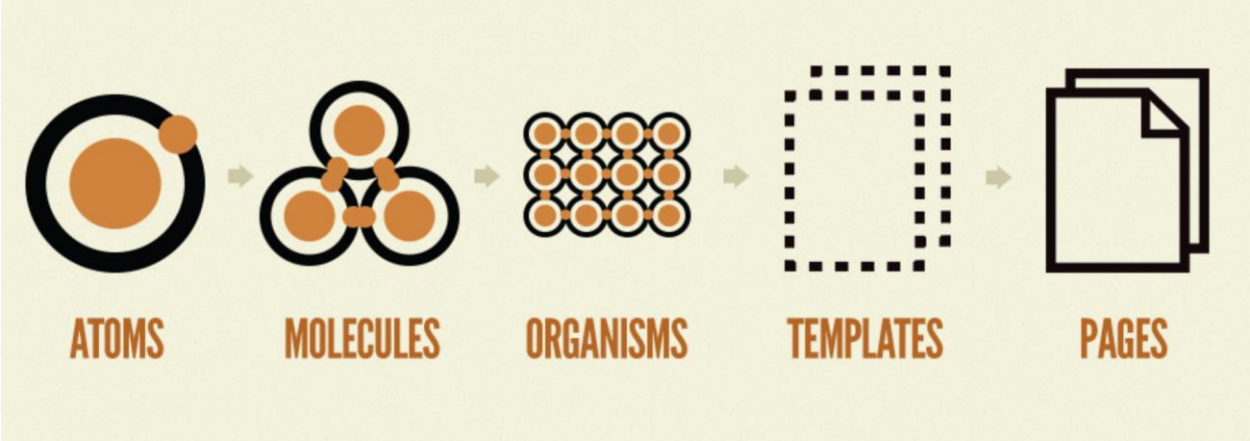

Рисунок 3.2.1 - Загальна архітектура Atomic design

Кожен атом у природному світі має свої унікальні властивості. Атом водню містить один електрон, тоді як атом гелію містить два. Ці властивості мають глибокий вплив на їх застосування (наприклад, вибух Гінденбурга був настільки катастрофічним, тому що дирижабль був заповнений надзвичайно горючим газом воднем проти інертного гелію). Таким же чином кожен атом інтерфейсу має свої унікальні властивості, такі як розміри зображення героя або розмір шрифту основного заголовка. Ці вроджені властивості впливають на те, як кожен атом повинен бути застосований до ширшої системи інтерфейсу користувача.

У контексті бібліотеки шаблонів, атоми демонструють всі ваші базові стилі на перший погляд, що може бути корисним посиланням, щоб продовжувати повертатися, коли ви розробляєте і підтримуєте вашу систему проектування. Але, як і атоми в природному світі, атоми інтерфейсу не існують у вакуумі і тільки дійсно оживають із застосуванням. Приклад застосування зображено на рисунку 3.2.2.

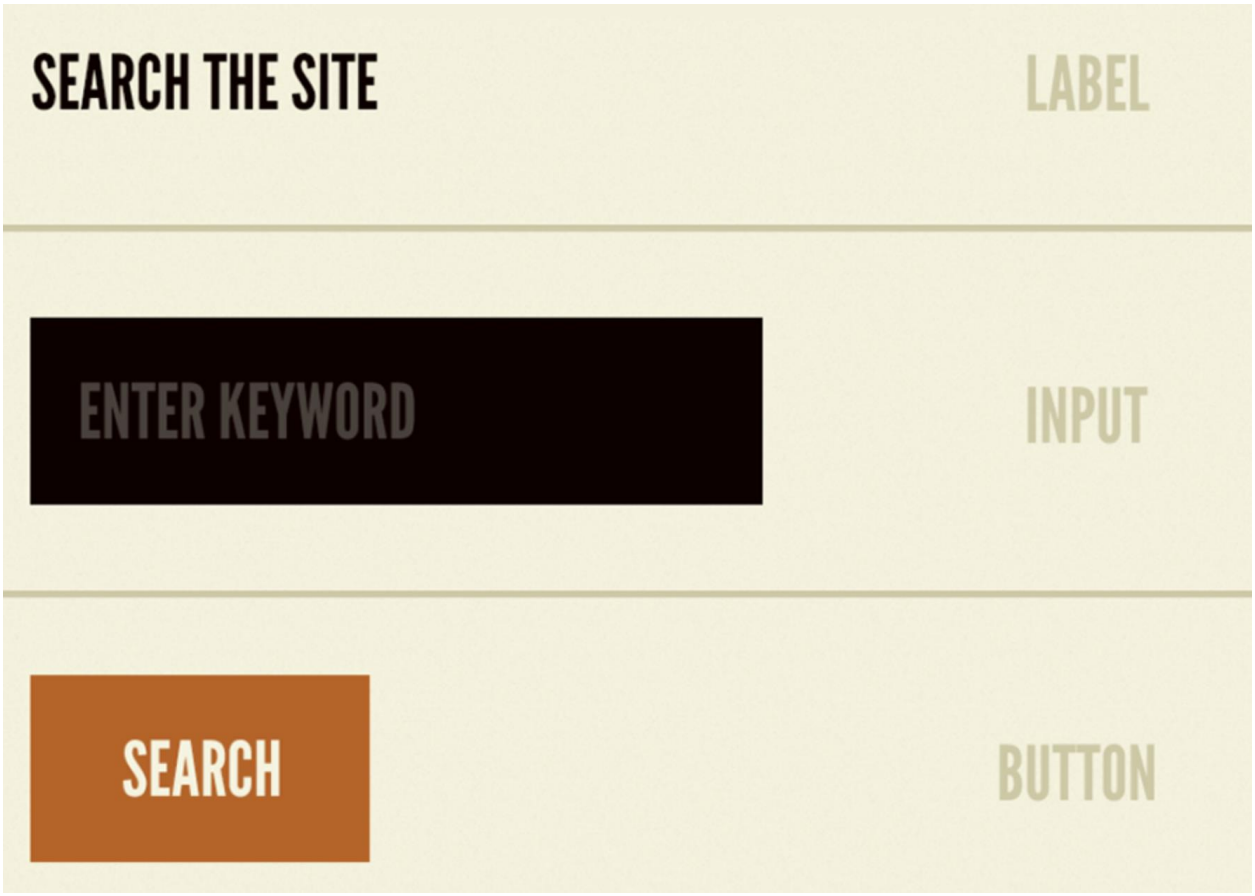

Рисунок 3.2.2 - Зображення атомів інтерфейсу

У інтерфейсах молекули є відносно простими групами елементів UI, що функціонують спільно як одиниця. Наприклад, мітка форми, вхідні дані пошуку та кнопки можуть об'єднуватися для створення молекули форми пошуку. Молекулу пошуку зображено на рисунку 3.2.3.

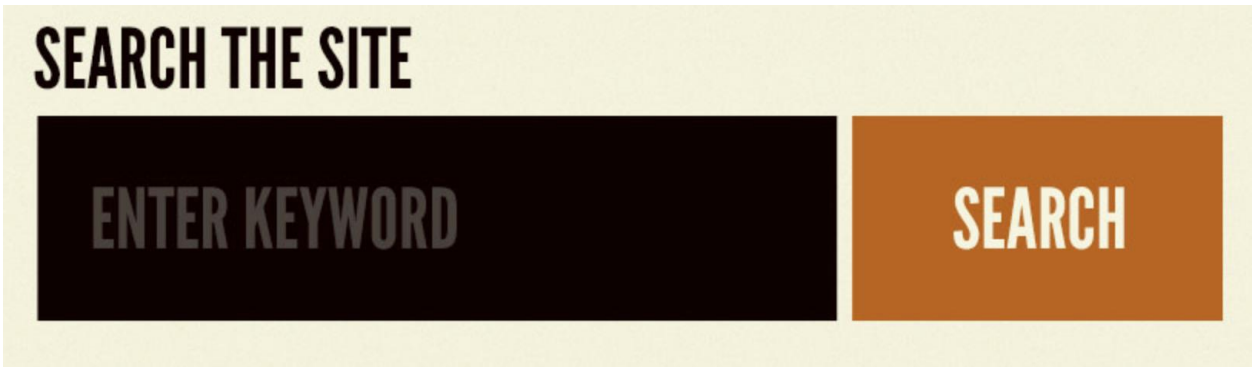

Рисунок 3.2.3 - Молекула пошуку

У поєднанні ці абстрактні атоми раптом мають мету. Атом етикетки тепер визначає вхідний атом. Натискання кнопки атома тепер подає форму. Результатом є простий, портативний, багаторазовий компонент, який може бути скинутий у будь-якому місці пошуку.

Тепер збирання елементів у прості функціонуючі групи - це те, що ми завжди робили для побудови інтерфейсів користувача. Але присвячення етапу методології атомного проектування цим відносно простим компонентам дає нам кілька ключових уявлень.

Створення простих компонентів допомагає дизайнерам та розробникам користувальницького інтерфейсу дотримуватися принципу єдиної відповідальності, старовинного принципу комп'ютерної науки, який заохочує менталітет «зробити одне і зробити це добре». Обтяження єдиного шаблону занадто складним робить програмне забезпечення громіздким. Таким чином, створення простих молекул інтерфейсу полегшує тестування, заохочує повторне використання і сприяє узгодженості в інтерфейсі.

Є прості, функціональні, багаторазові компоненти, які ми можемо поставити в більш широкий контекст.

Організми є відносно складними UI компонентами, що складаються з груп молекул та / або атомів та / або інших організмів. Ці організми формують окремі ділянки інтерфейсу.

Давайте повернемося до нашої молекули форми пошуку. Форма пошуку може часто знаходитися в заголовку багатьох веб-досвіду, тому давайте покладемо цю молекулу форми пошуку в контекст організму заголовка, зображеного на рисунку 3.2.4.

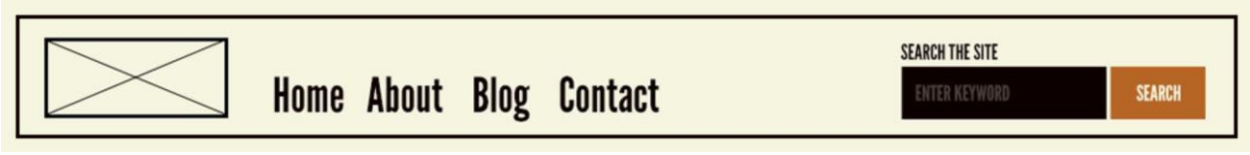

Рисунок 3.2.4 - Об’єднанні молекули та атоми в організм

Заголовок утворює окремий розділ інтерфейсу, хоча він містить декілька менших фрагментів інтерфейсу з власними унікальними властивостями та функціональністю.

Організми можуть складатися з подібних або різних типів молекул. Орган заголовка може складатися з різнорідних елементів, таких як зображення логотипу, основний список навігації та форма пошуку. Ми бачимо ці типи організмів майже на кожному сайті, який ми відвідуємо.

Створення від молекул до більш складних організмів надає дизайнерам і розробникам важливе значення контексту. Організми демонструють ті менші, простіші компоненти в дії і служать різними моделями, які можна використовувати знову і знову. Організм продуктової сітки може бути використаний в будь-якому місці, де повинна бути відображена група продуктів, від списків категорій до результатів пошуку до відповідних продуктів.

Тепер, коли ми маємо організми, визначені в нашій системі проектування, ми можемо порушити нашу аналогію хімії і застосувати всі ці компоненти до того, що нагадує веб-сторінку.

Мова атомів, молекул і організмів несе в собі корисну ієрархію, щоб ми свідомо будували компоненти наших систем проектування. Але в кінцевому підсумку ми повинні перейти до мови, яка є більш прийнятною для нашого кінцевого результату, і має більше сенсу для наших клієнтів, босів і колег.

Шаблони - це об'єкти на рівні сторінки, які розміщують компоненти в макеті та формулюють базову структуру дизайну. Щоб побудувати наш попередній приклад, ми можемо взяти заголовок і застосувати його до шаблону домашньої сторінки, яка зображена на рисунку 3.2.5.

Цей шаблон домашньої сторінки відображає всі необхідні компоненти сторінки, що працюють разом, що забезпечує контекст для цих відносно абстрактних молекул і організмів.

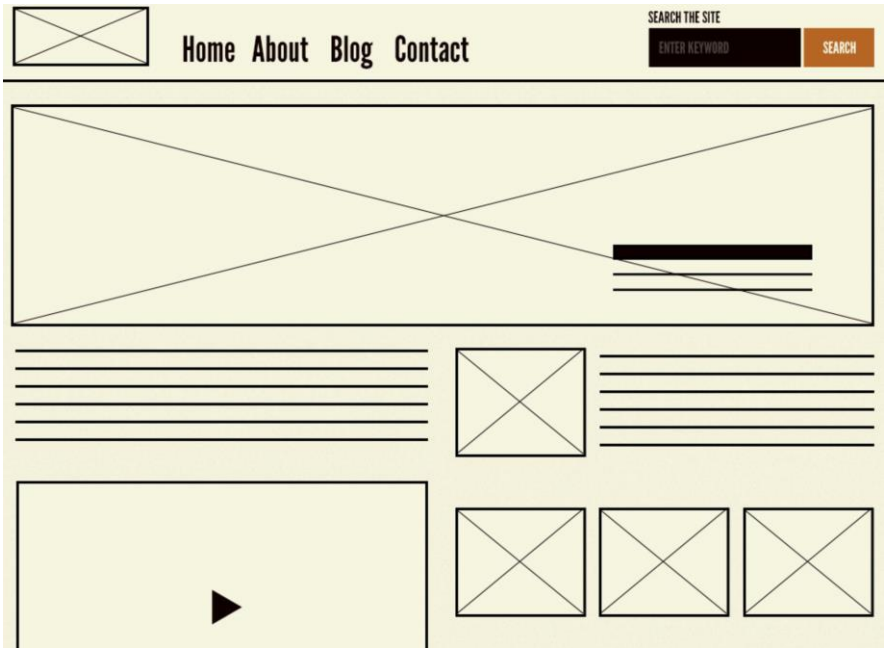

Рисунок 3.2.5 - Шаблон домашньої сторінки

При розробці ефективної системи проектування, важливо продемонструвати, як компоненти виглядають і функціонують разом у контексті макета, щоб довести, що частини додаються до добре функціонуючого цілого.

Іншою важливою характеристикою шаблонів є те, що вони орієнтуються на структуру вмісту, яка лежить в основі, а не на остаточному змісті сторінки. Системи проектування повинні враховувати динамічний характер вмісту, тому дуже корисно формулювати важливі властивості компонентів, таких як розміри зображень і довжини символів для заголовків і текстових фрагментів.

Визначивши скелет сторінки, ми можемо створити систему, яка може враховувати різноманітність динамічного вмісту, забезпечуючи при цьому необхідні огородження для типів вмісту, які заповнюють певні шаблони дизайну.

Сторінки - це конкретні приклади шаблонів, які показують, як виглядає користувальницький інтерфейс із реальним представницьким вмістом. На основі нашого попереднього прикладу, ми можемо взяти шаблон домашньої сторінки і вилити репрезентативний текст, зображення та медіа в шаблон, щоб показати реальний вміст у дії, рисунок 3.2.6.

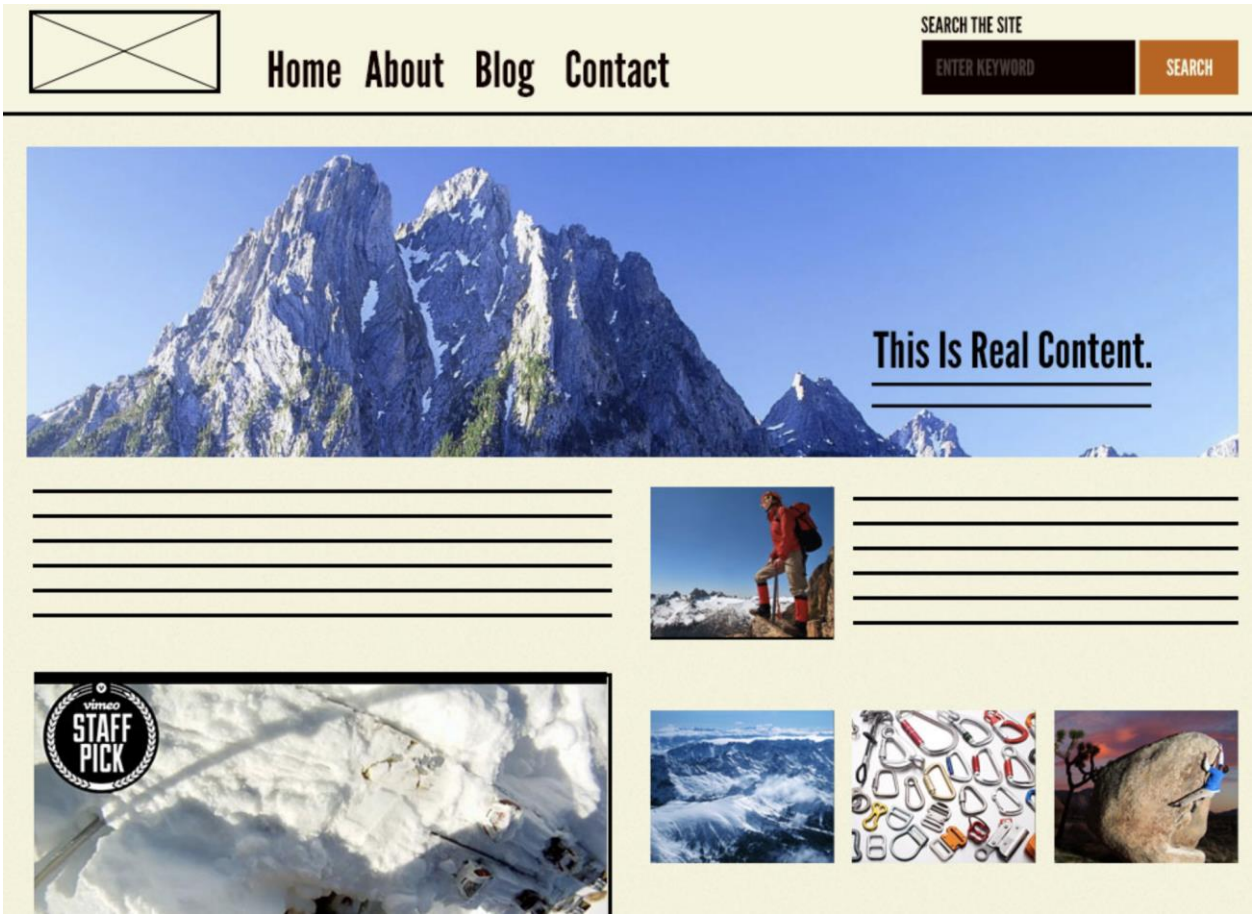

Рисунок 3.2.6 - Шаблон домашньої сторінки

Етап сторінки є найбільш конкретним етапом атомного проектування, і це важливо з деяких досить очевидних причин. Зрештою, це те, що користувачі зможуть побачити і взаємодіяти, коли вони відвідають ваш досвід. Це те, що ваші зацікавлені сторони підпишуть. І саме тут ви бачите всі ці компоненти, щоб сформувати гарний і функціональний інтерфейс користувача.

Окрім демонстрації остаточного інтерфейсу, якого користувачі бачать, сторінки є важливими для перевірки ефективності базової системи проектування. Саме на етапі сторінки ми можемо поглянути на те, як всі ці моделі затримуються, коли реальний вміст застосовується до системи проектування.

Ми повинні створити системи, які встановлюють багаторазові шаблони проектування, а також точно відображають реальність змісту, який ми розміщуємо всередині цих моделей.

Сторінки також забезпечують місце для формулювання варіацій шаблонів, що є надзвичайно важливим для створення надійних та надійних систем проектування. Ось лише кілька прикладів варіацій шаблонів:

– Користувач має один елемент у кошику для покупок, а інший користувач має десять елементів у кошику;

– Інформаційна панель веб-програми зазвичай показує нещодавню активність, але цей розділ припиняється для користувачів, які вперше використовуються;

– Один заголовок статті може складатися з 40 символів, а інший заголовок статті може складатися з 340 символів;

– Користувачі з правами адміністратора можуть бачити додаткові кнопки та параметри на інформаційній панелі порівняно з користувачами, які не є адміністраторами.

У всіх цих прикладах базові шаблони однакові, але користувальницькі інтерфейси змінюються для відображення динамічного характеру вмісту. Ці зміни прямо впливають на те, як будуються основні молекули, організми і шаблони. Тому створення сторінок, які враховують ці варіації, допомагає нам створювати більш стійкі системи проектування.

Отже, це атомний дизайн. Ці п'ять різних етапів одночасно працюють разом для створення ефективних систем дизайну інтерфейсу користувача. Підсумовуючи атомний дизайн у двох словах:

– Атоми - це елементи інтерфейсу, які не можуть бути розбиті далі і служать елементарними будівельними блоками інтерфейсу;

– Молекули є колекціями атомів, які утворюють відносно прості компоненти UI;

– Організми є відносно складними компонентами, які утворюють дискретні розділи інтерфейсу;

– Шаблони розміщують компоненти в межах макета і демонструють структуру змісту, що лежить в основі дизайну;

– Сторінки застосовують реальний вміст до шаблонів і змінних варіантів, щоб продемонструвати кінцевий інтерфейс і перевірити стійкість системи проектування.

Одним з найбільших переваг атомного дизайну є можливість швидкого перемикання між абстрактним і конкретним. Ми можемо одночасно бачити наші інтерфейси, розбиті на їх атомні елементи, а також бачити, як ці елементи поєднуються разом для формування нашого кінцевого досвіду.

Але для нашого застосунку ця архітектура не підходить, тому що вона розрахована на більш складні проектні застосунки, в команді яких працює більше ніж 20 розробників.

Domain-driven design (DDD) - метод розробки програмного забезпечення, запропонований Еріком Евансом, включає стратегічні, філософські, тактичні та технічні елементи і пов'язаний з багатьма конкретними практиками.

Суть DDD є дуже простою: захопити модель домену в доменних термінах, вставити модель в код і захистити її від колізії. Хоча цей рівень розуміння є трохи неглибоким, він все ще є корисним - і він може бути достатньо освіжаючим, щоб прийняти рішення.

DDD має стратегічне значення; тому настільки багато компаній з надзвичайно складними доменами покладаються на нього, щоб виробляти програмне забезпечення, яке може швидко розвиватися разом із бізнесом. Підкреслюючи фундаментальний принцип DDD: використовуйте доменну термінологію всюди; зробити його повсюдним.

Характеристики сильної доменної моделі:

– Бути узгодженим з моделлю, стратегіями та процесами компанії;

– Бути ізольованими від інших доменів;

– Бути вільно залежними від шарів програми на будь-якій стороні шару домену;

– Бути багаторазовим, щоб уникнути дублювання моделей;

– Бути абстрактним і чисто розділеним шаром, щоб створити легше обслуговування, тестування і версії;

– Мінімальна залежність від інфраструктурних рамок, щоб уникнути вичерпання цих рамок і жорсткої зв'язку між зовнішніми рамками.

Переваги DDD:

– Бізнес-потреби орієнтовані - з дизайном, керованим доменами, кожен користується однією мовою та термінами, а команда ділиться моделлю. Розробники краще спілкуються з бізнес-командою, і робота є більш ефективною, коли йдеться про створення рішень для моделей, які відображають, як бізнес працює, а не як працює програмне забезпечення;

– Загальний набір термінів і визначень, що використовуються всією командою - команди знаходять спілкування набагато простіше під час циклу розробки, оскільки з самого початку вони зосереджуються на створенні повсюдної мови, яка є спільною для обох сторін (експерти з розвитку та бізнесу). Мова пов'язана з доменною моделлю проекту, а технічні аспекти називаються простими термінами, які всі розуміють;

– Відстеження стало простіше - якщо кожна людина використовує одну й ту ж термінологію, то буде досить просто відстежувати виконання вимог;

– Кращий код - з DDD ви в кінцевому підсумку з більш читабельний код і менше дублювання;

– Дотримуючись підхід Agile, який є ітеративним і інкрементним, DDD роз'яснює ментальні моделі експертів домену в корисну модель для бізнесу;

– Гарна архітектура програмного забезпечення - всі команди здатні зрозуміти, де важливі деякі інтеграції і чому;

– Комунікаційні підрахунки - DDD дуже зручно, коли йдеться про допомогу команді у створенні загальної моделі. Команди з боку бізнесу та з боку розробника можуть використовувати цю модель для обміну інформацією про бізнес-вимоги, об'єкти даних та моделі процесів.

– Збалансоване застосування - з DDD, будуєте навколо концепцій домену і навколо того, що експерти домену консультують. Це означає, що розроблені програми дійсно представлятимуть те, що потребує домен, замість того, щоб отримувати додаток, орієнтований лише на UX / UI, забуваючи про інші вимоги. Це допомагає створити збалансований продукт, який підходить користувачам / аудиторіям певного домену.

– Гнучка модель - DDD обертається навколо концепцій об'єктно-орієнтованого проектування. Це означає, що практично все в моделі домену базується на об'єкті і тому буде модульним і інкапсульованим, що дозволяє системі регулярно і постійно змінюватися і вдосконалюватися.

3.3 Архітектура веб-застосунка

В застосунку використовується DDD архітектура тому що вона гарно масштабується для декількох чоловік в команді та гнучка у розробці. Ви можете побачити її на рисунку 3.3.1.

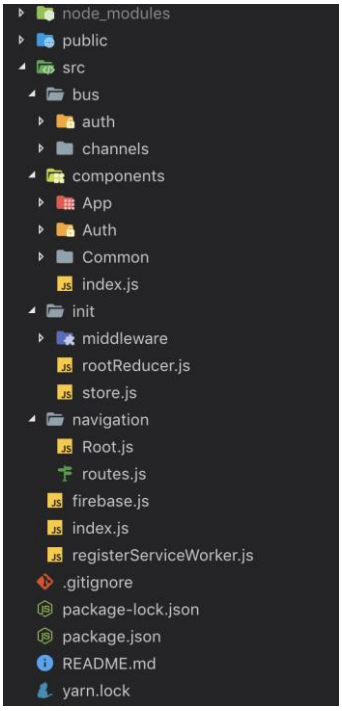

Рисунок 3.3.1 - Структура застосунку корпоративний чат

Принцип побудови шинної архітектури:

Додаток розбивається на доменні частини, логіка взаємодії з API і асинхронні операції за винятком роботи з firebase винесені в окремі частини програми.

В папці bus знаходяться папки окремих сторінок, які відповідають за роботу з асинхронними запитами, даними в redux, і екшени, рисунок 3.3.2.

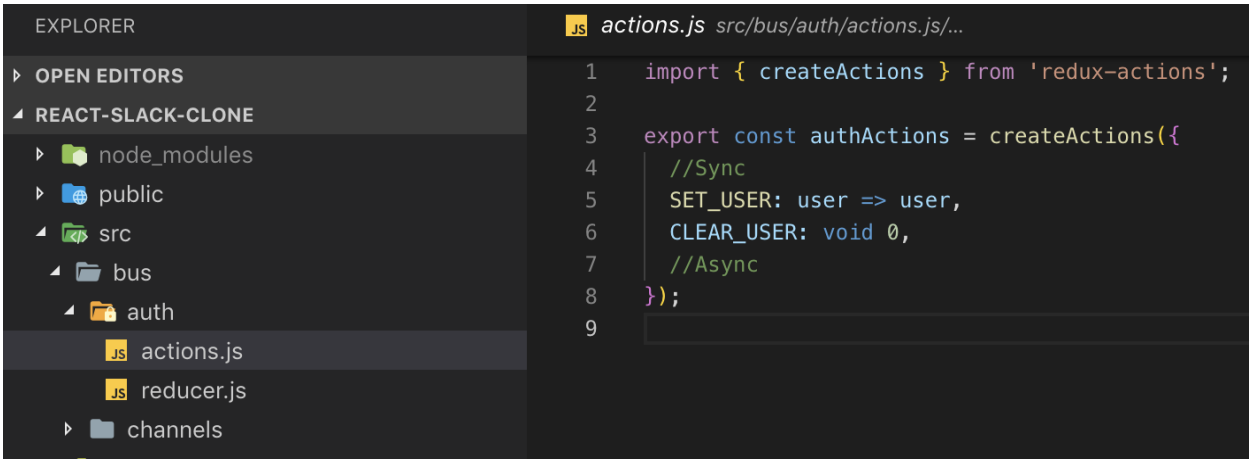

Рисунок 3.3.2 - Модель модулю bus

В папці Components, є окремий модуль Common, в нього винесені всі компоненти, які будуть перевикористовуватися надалі, такі як кнопки, лоадер і т.д. У цій папці так само все розбите по доменах, одна сторінка - окремий домен для неї. Це можна побачити на рисунку 3.3.3.

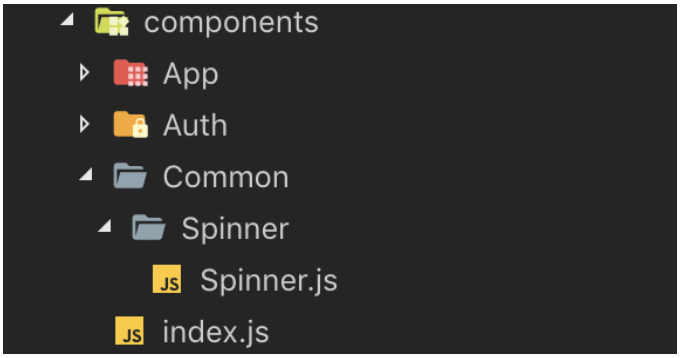

Рисунок 3.3.3 - Модель модулю Components

В папці init зберігається ініціалізація всіх наших middlewares, reducers і store, зображено на рисунку 3.3.4.

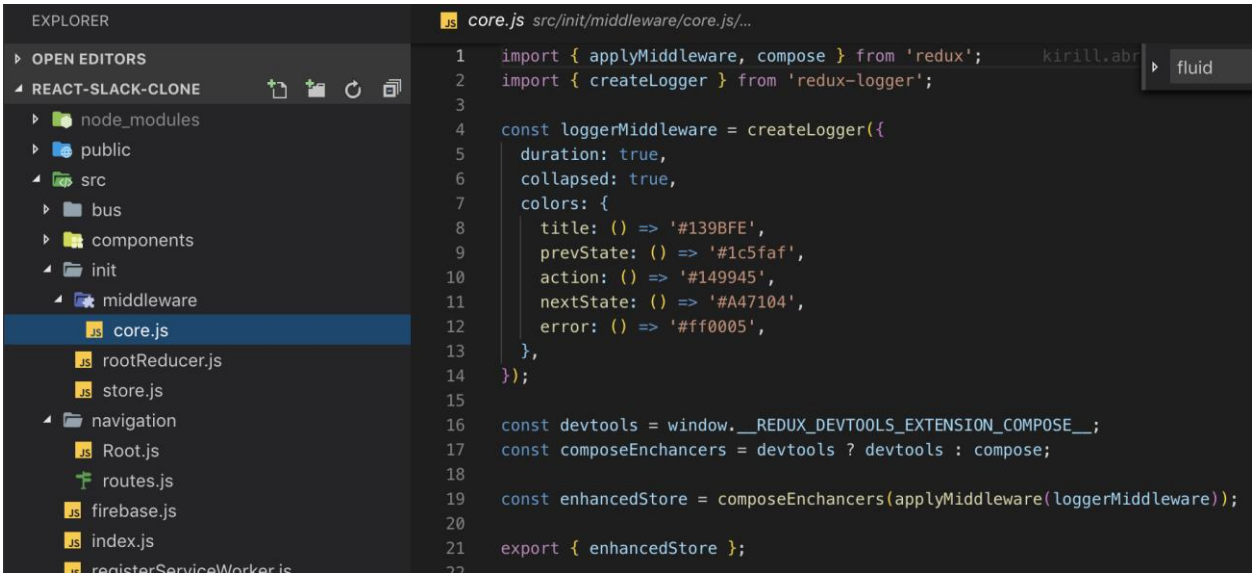

Рисунок 3.3.4 - Модель модулю init

Навігація зроблена у Root файлі, в якому є приватні та публічні сторінки, якщо ви не є зареєстрованим користувачем в застосунку, то вам будуть доступні тільки публічні сторінки, в зворотному випадку - приватні, рисунок 3.3.4.

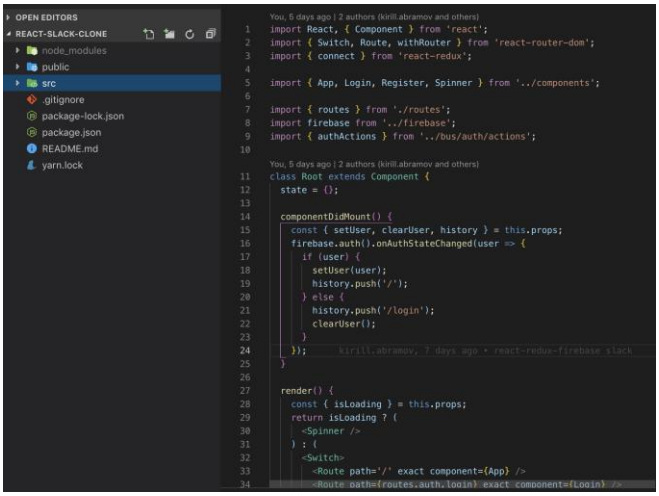

Рисунок 3.3.4 - Компонент навігації в застосунку

routes.js - це заморожений об'єкт в якому зберігаються всі шляхи нашого SPA застосунку за доменним стилем. Цей модуль можна побачити на рисунку 3.3.5.

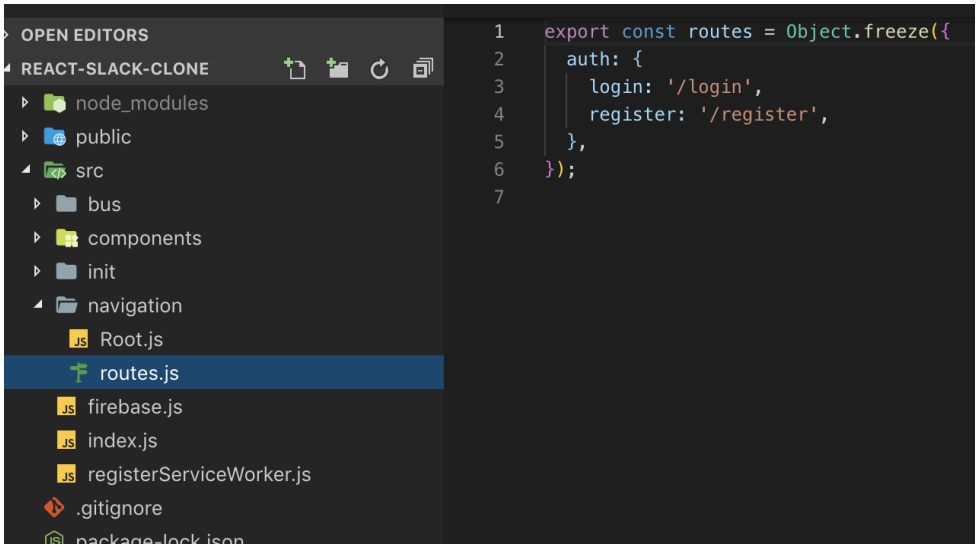

Рисунок 3.3.5 - Модуль шляхів застосунку

firebase.js - в цьому файлі проходить ініціалізація фаєрбейз, рисунок 3.3.6

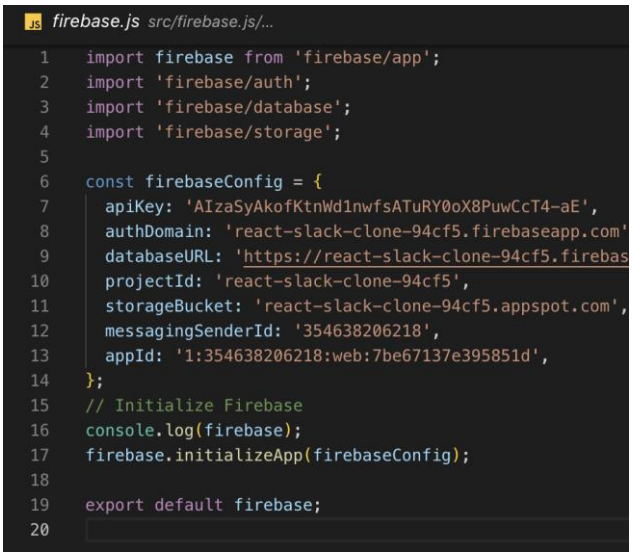

Рисунок 3.3.6 - Ініціалізація firebase

3.4 Модулі збереження та обробки даних

Сховище даних додатка є спеціальним незмінним об'єктом з бібліотеки Redux. Воно бере на себе такі завдання:

- Містить стан додатки (application state);

- Надає доступ до стану за допомогою методу getState ();
- Надає можливість поновлення стану за допомогою методу dispatch (action);
- Реєструє слухачів (listeners) с допомогою методу subscribe (listener).

Додаток не може безпосередньо змінити стан сховища. замість цього, необхідно передати екшен (англ. action - дія), тобто об'єкт, описує, що сталося, щоб висловити намір змінити стан в сховище. Безпосередньо зміною стану займаються редьюсери - функції, які обробляють екшени і можуть вносити зміни в стан. Причому, редьюсери не повинні змінювати існуючий стан. Вони працюють з його копією, повертають її і після цього відбувається перезапис стану.

Філософія Redux має на увазі наявність єдиного сховища в додатку, тому для поділу логіки використовується кілька рівнів вкладеності об'єкта.

Ці принципи не є фіксованими правилами або буквальними висловлюваннями щодо реалізації Redux. Скоріше, вони формують заяву про наміри щодо використання Redux. Reducer можна побачити на рисунку 3.4.1

```
import { handleActions } from 'redux-actions';

import { authActions } from './actions';

const initialState = {      kirill.abramov, 7 days ag
  currentUser: null,
  isLoading: true,
};

export const authReducer = handleActions(
  {
    [authActions.setUser]: (state, { payload }) => {
      return {
        ...state,
        currentUser: payload,
        isLoading: false,
      };
    },
    [authActions.clearUser]: state => {
      return {
        ...state,
        isLoading: false,
      };
    },
  },
  initialState,
);
```

Рисунок 3.4.1 - Редьюсер авторизації

3.5 Діаграма прецендентів

Діаграма прецедентів - діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні. Дана діаграма призначена для визначення функціональних вимог до програмного продукту.

Базовими елементами діаграми прецедентів є актори і прецеденти. Актор - роль, яку відіграють зовнішні сутності. Прецедент - послідовності дій, які система або інша сутність можуть виконувати в процесі взаємодії з акторами. Були виділені актори: адміністратор і користувач.

Діаграма роботи з профілем користувача представлена на рисунку 3.5.1. Користувач може переглядати і редагувати свій профіль, переглядати проекти, повідомлення, організацію.

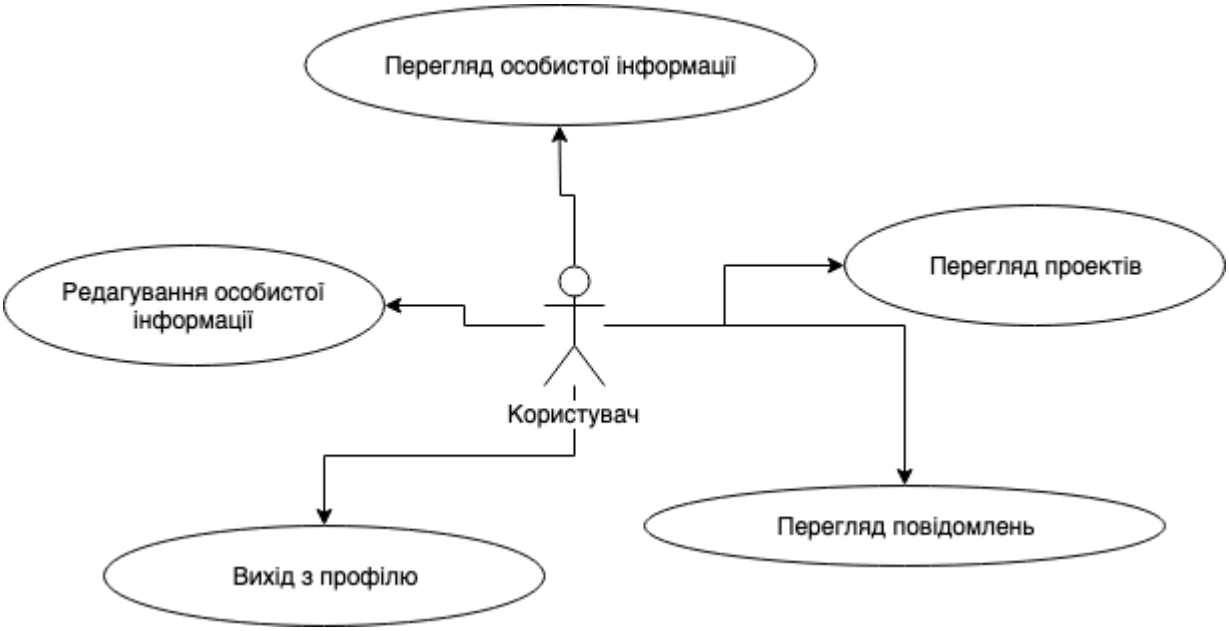

Рисунок 3.5.1 - Use case діаграма роботи з профілем користувача

Діаграма роботи адміністратора представлена на рисунку 3.5.2. Адміністратор веде облік існуючих користувачів, може редагувати список посад,

користувачів, а так само може редагувати існуючі проекти, змінювати статус користувачів.

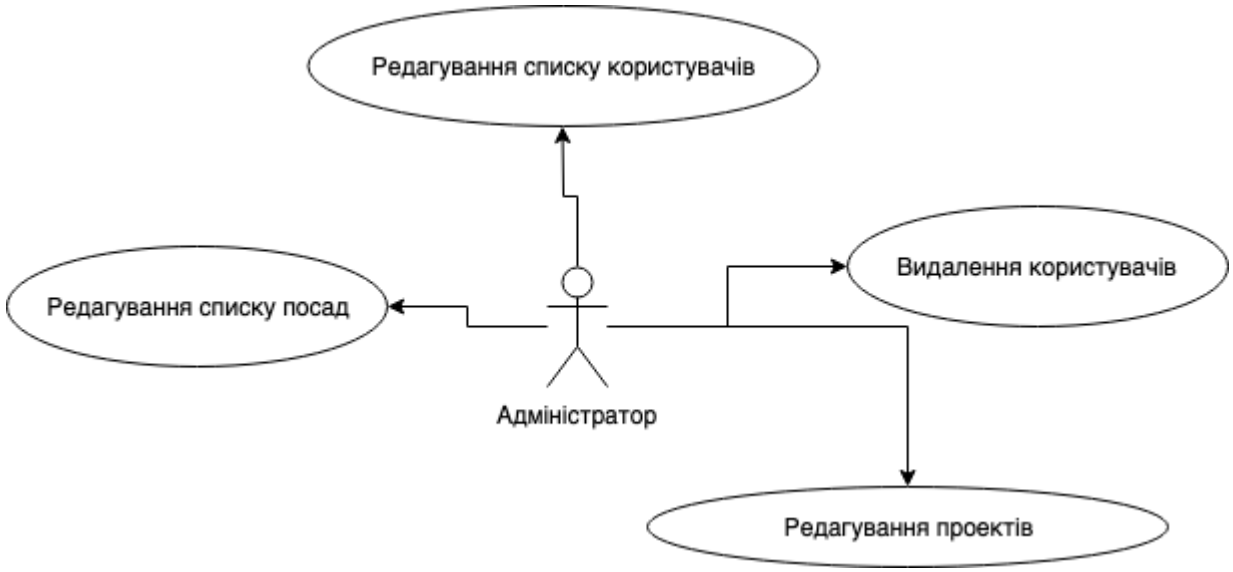

Рисунок 3.5.2 - Use case діаграма роботи адміністратора

Діаграма роботи з проектами представлена на рисунку 3.5.3. Користувач може створювати, редагувати проекти, додавати учасників, документи, робочі групи проекту, створювати, редагувати бесіди робочих груп.

Рисунок 3.5.3 - Use Case діаграма роботи з проектами

Діаграма роботи з повідомленнями представлена на рисунку 3.5.4. Користувач може створювати бесіди, писати, додавати учасників. Бесіди можуть бути в робочих групах і просто між користувачами

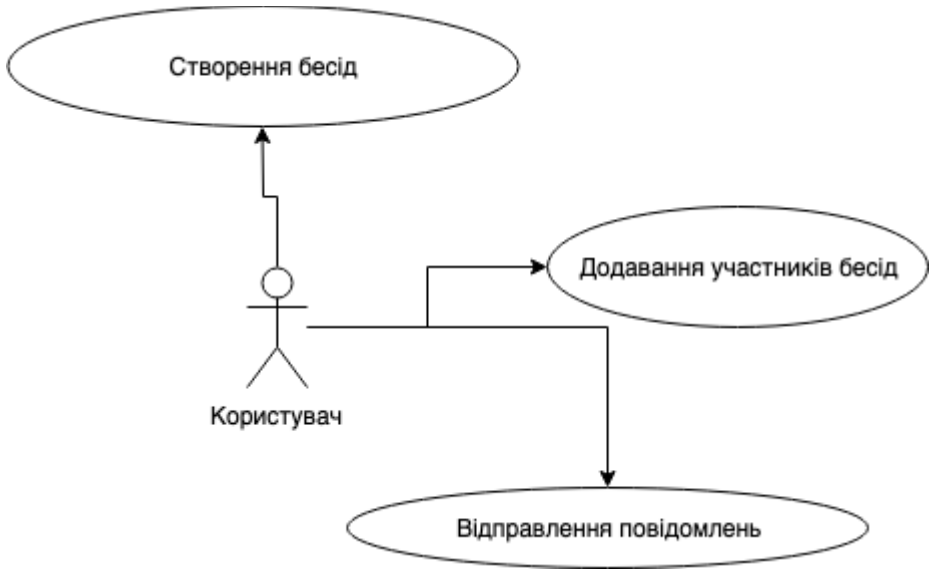

Рисунок 3.5.4 - Use Case діаграма роботи з повідомленнями

Гранулярність випадку використання відноситься до способу, в якому інформація організована в специфікаціях випадку використання, і в якійсь мірі, рівень деталізації, на якому вони написані. Досягнення належного рівня деталізації випадків використання полегшує зв'язок між зацікавленими сторонами та розробниками та покращує планування проекту.

Висновок до розділу

У даному розділі було розглянуто архітектуру односторінкового веб-застосунку, яка використовується. Її переваги між іншими архітектурами, також були розглянуті Use Case діаграми для роботи з різними ролями користувачів, повідомлень та груп.

Також були розглянуті та детально описані всі модулі застосунка.

4 РЕАЛІЗАЦІЯ ОДНОСТОРІНКОВОГО ВЕБ-ЗАСТОСУНКУ

4.1 Налаштування необхідних модулів для початку роботи з застосунком

Для початку роботи з веб-застосунком необхідно встановити на свій ПК модулі, які дозволять запустити застосунок.

4.1.1 Встановлення Node.js

NVM - це скрипт bash, який використовується для керування кількома випущеними версіями Node.js. Вона дозволяє виконувати операції, такі як інсталяція, видалення, перемикання версій і т.д.

Npx - це засіб, який допомагає завершити роботу з використанням пакунків з реєстру npm - так само, як npm робить його надзвичайно простим у встановленні та керуванні залежностями, розміщеними в реєстрі, npx дозволяє легко використовувати інструменти CLI та інші виконувані файли, розміщені на реєстру. Для того щоб встановити Nodejs необхідно прописати в консолі команди нижче.

```
sudo apt-get install curl python-software-properties
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

4.2 Опис роботи застосунку

Робота в системі починається з завантаження головної сторінки. Якщо користувач не авторизований, йому пропонується увійти в систему або зареєструватися.

Для початку роботи з веб-застосунком необхідно встановити на свій ПК модулі, які дозволять запустити застосунок.

Для початку роботи з додатком користувачеві необхідно пройти реєстрацію. Для цього користувачеві необхідно заповнити його основні дані. Сторінка реєстрації користувача представлена на рисунку 4.2.1.

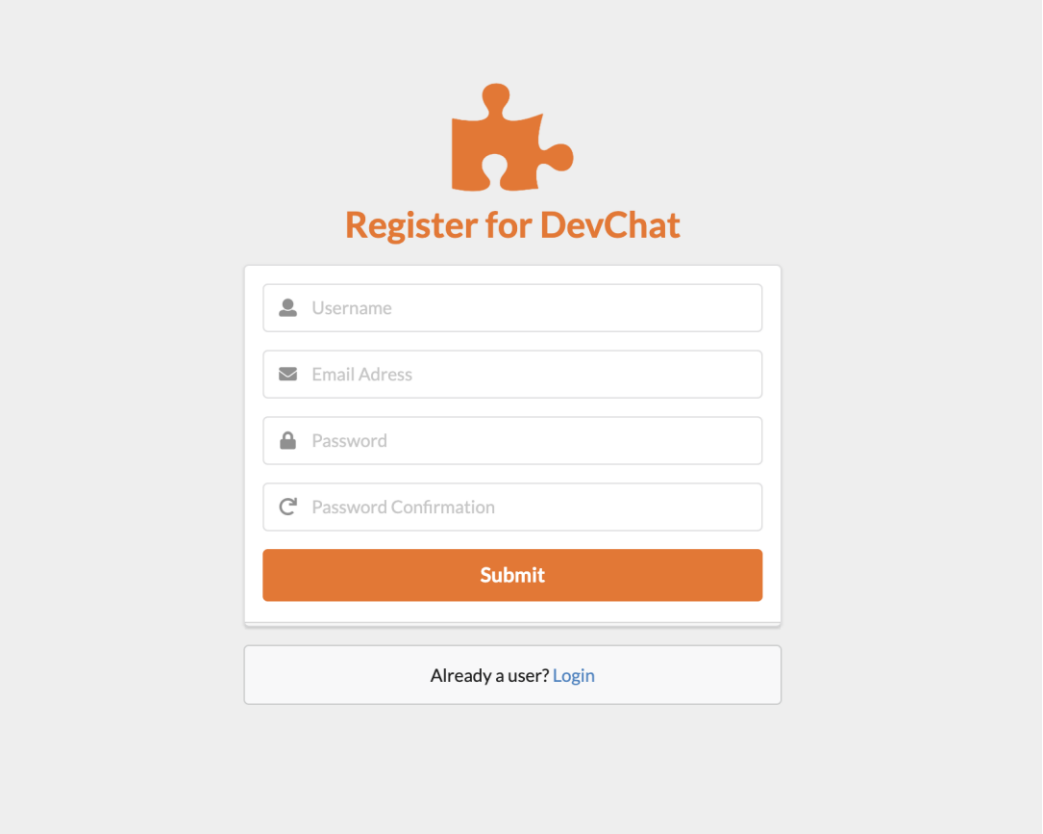

Рисунок 4.2.1 - Сторінка реєстрації користувача

На рисунку 4.2.2 представлена форма авторизації користувача в системі. Для цього йому необхідно ввести логін і пароль.

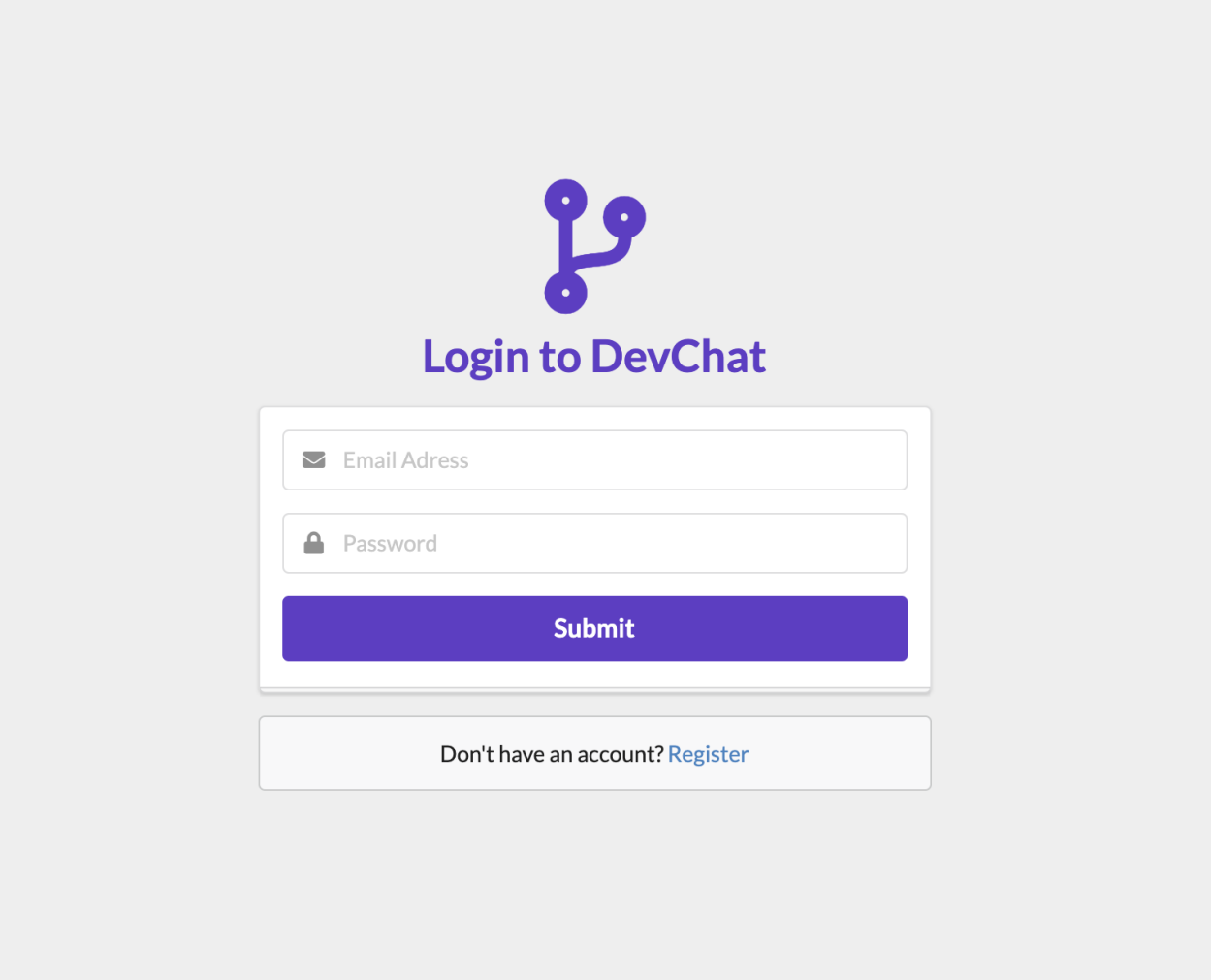

Рисунок 4.2.2 - Сторінка входу в систему

Після успішного входу в систему користувач попадає на головну приватну сторінку, яка представлена на рисунку 4.2.3.

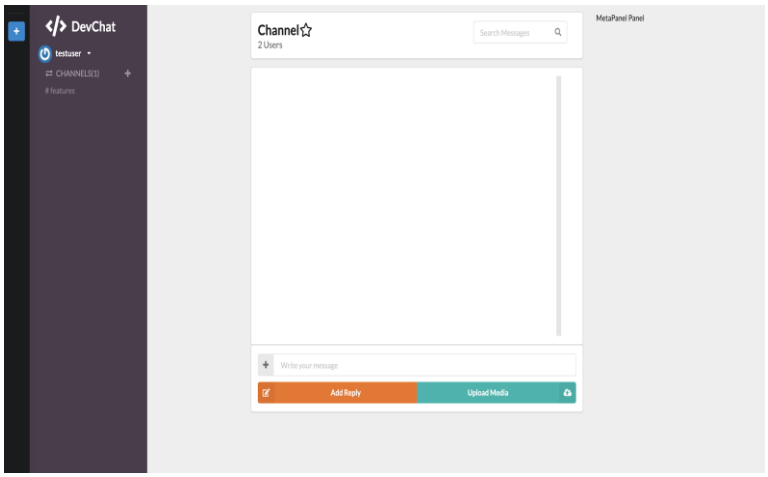

Рисунок 4.2.3 - Головна приватна сторінка

Для створення нового каналу необхідно нажати на кнопку “+”, яка знаходиться біля списку всіх каналів на головній сторінці. Після цього відкриється модальне вікно яке зображене на рисунку 4.2.4, необхідно заповнити такі поля як назва каналу, та його опис.

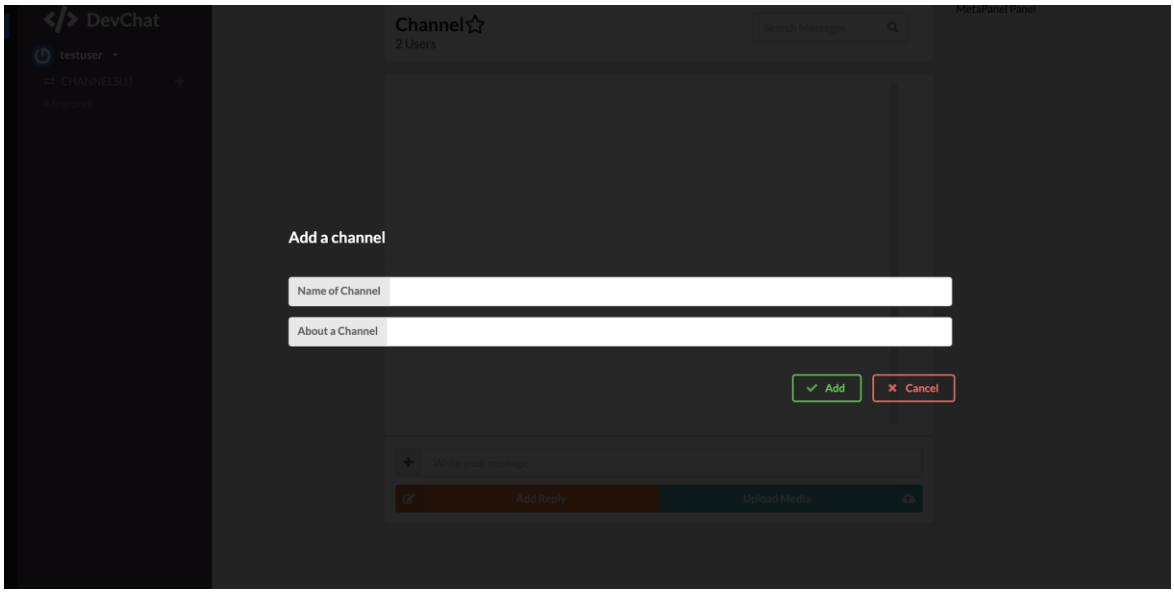

Рисунок 4.2.4 - Створення нового каналу

Висновок до розділу

У даному розділі було описано як розгорнути проект у себе на ПК, взаємодія користувача з застосунком, створення нового користувача, його вхід в систему, вікна каналів, які доступні певному користувачу, створення каналів.

ВИСНОВКИ

В результаті виконання бакалаврської роботи були сформовані вимоги до системи, вивчені інформація про дану предметної області, технології, за допомогою яких спроектована необхідна система, вивчені і проаналізовані аналоги, що розробляється. Були спроектовані база даних і сама система.

Результатом є розробленне веб-застосування "Односторінкове застосування для корпоративної переписки". Застосування дозволяє ефективно взаємодіяти між собою учасникам робочого процесу, що дозволяє оптимізувати роботу як окремого користувача, так і цілої компанії.

Проаналізовано та обрано підходи та типи тестування системи, описано процес тестування, виконано тестування системи. Описано вимоги до середовища, а саме мінімальні системні вимоги.

Так само були обрані і використані технології, які дозволили спростити розробку програмного коду і поліпшити його якість.

За допомогою бібліотеки Mocha було проведено тестування основної частини програми. Це дозволило виявити ряд помилок в роботі програми і виправити їх. Подальший розвиток цього додатка буде направлено на поліпшення якості, додавання нових функцій, що дозволяють розширити можливості для навігації з використанням створених моделей.

ПЕРЕЛІК ПОСИЛАНЬ

1. React documentation [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://reactjs.org/docs/getting-started.html.
2. Redux saga magic in react [Електронний ресурс] – Режим доступу до ресурсу:https://medium.com/@js_tut/the-saga-continues-magic-in-react 44da8d134285.
3. Redux saga tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://redux-saga.js.org/docs/introduction/BeginnerTutorial.html.
4. Redux tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/439104/.
5. Redux saga tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/351168/
6. Redux actions documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redux-actions.js.org/api/createaction.
7. Лекция Redux saga [Електронний ресурс] – Режим доступу до ресурсу: https://gist.github.com/nikneroz/c96d48563549abbe997abef5433fb111
8. Redux documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/.
9. Mocha documentation [Електронний ресурс] – Режим доступу до ресурсу: https://mochajs.org/.
10. DDD Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Domain-driven_design.
11. Node js documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/.
12. Firebase: прощання з ілюзіями [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/livetyping/blog/320040/.
13. Javascript Air [Електронний ресурс] – Режим доступу до ресурсу: https://javascriptair.com/.

14. Firebase documentation [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Firebase

15. FACTORY FUNCTIONS AND THE MODULE PATTERN [Електронний ресурс] – Режим доступу до ресурсу: https://www.theodinproject.com/courses/javascript/.

16. React vs. Angular: The complete comparison [Електронній ресурс] – Режим доступу до ресурсу: https://programmingwithmosh.com/react/react-vs-angular/

ДОДАТОК А.

ДОДАТОК Б.

ДОДАТОК В.